ADAS

By Steven A. Bailey

NASA
June 29, 1998

Table of Contents

Introduction

The ADAS (Airborne Diode Array Spectrometer) is a small 256 channel visible spectrometer covering 400 to 720 nm. The main optics consists of a 4 inch Newtonian telescope followed by a fiber-optic light guide which feeds a diffraction grating. The spectrometer sensor is a solid-state diode array in the form of a 22 pin DIP. It rests on a PC board which is mounted at the focal point below the diffraction grating. See Figure 1 for simplified view of the system.

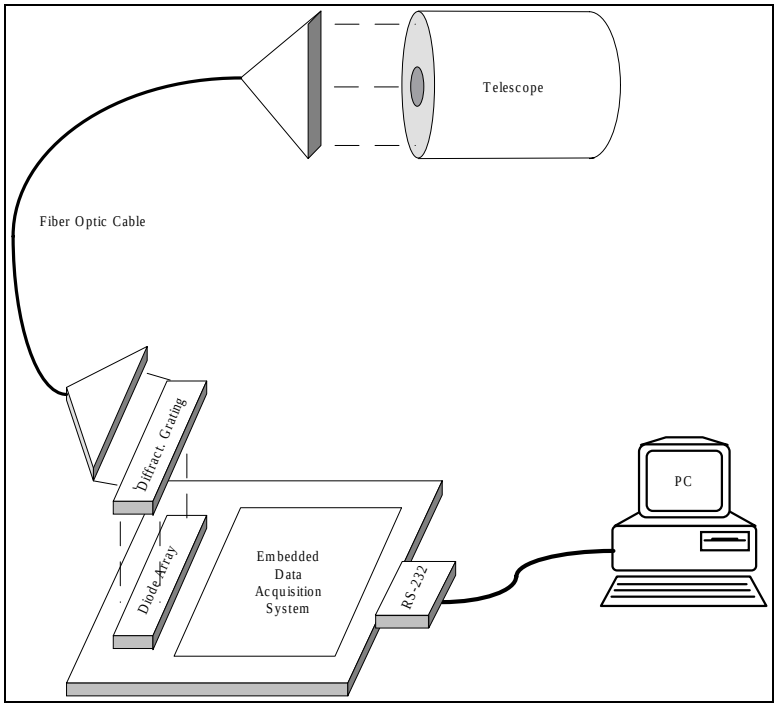

Figure 1

ADAS is a system that was initially developed in the mid-1980s. It started as a commercial spectrometer system that included the sensor, optics, and electronics. Its electronics were modified and the system performed reasonably well. Since then, ADAS has evolved over the years to its current state which is a commercial sensor board surrounded by custom built optics. This version has worked quite well, but has some inherent limitations and design flaws.

The current system contains all analog circuitry on a PC board. The signals that drive or 'clock' this system must reside external to the board. Also, analog to digital conversion must be external to the board. These two factors cause an equal number of problems. An external computer (like a PC) must be available to both 'clock' the sensor board, perform the AD conversion, and record data. Due to stringent 'clocking' demands, this requires a single PC which is not available for any other duties. In fact, the PC causes a measurable amount of 'clock' jitter to the sensor. This results in inconsistent integration times

between sensor channels. Long cable lengths between the sensor board and the PC also cause additional noise on the analog video line.

Purpose

My purpose on this project was to start from scratch and design a new autonomous sensor board. It would contain the sensor, a high speed 12-bit ADC, a microcontroller, and all supporting analog and digital circuitry. This system would generate all clocking signals itself, perform AD conversions, and buffer data. Digital data would pass from the sensor board to any external computer that had an RS232 port.

This design would remove the timing demands from an external PC. This way, more than one external spectrometer could be tied to a single PC provided there are enough RS232 ports. There are plans to plug 3 of these spectrometers into a single PC running LINUX fitted with a multichannel RS232 board.

This design also removes the analog signal noise that was a problem in all previous designs. Since all analog conditioning, amplification, and digitization are onboard the new sensor board, there is no opportunity for noise interjection into the raw signal.

Basic Design

The design of the hardware is straight forward. Essentially, a microcontroller is the heart of the system. It resides on the new spectrometer board and basically has 3 functions.

First of all, this microcontroller 'clocks' the diode array at some given rate and then triggers the ADC to perform a 12-bit conversion. This cycle occurs 256 times…once for each channel of the sensor. This data is saved in a 2K static ram (sram).

Data that is being buffered in sram is exported or transmitted out the RS232 port at a rate of 38,400 baud. This rate supports a maximum video frame rate of 5 spectra per second.

Finally, single byte commands are received from an external PC. These commands instruct the microcontroller to perform things like open or close the shutter, record data at 'some' frame rate, change the exposure time, etc. See Figure 2 for block diagram of the spectrometer board.

Electronics

A 16C65A Microchip microcontroller was chosen for several reasons. First, it has a very short instruction time (200ns). This short time is needed because the diode array and

ADC would be 'clocked' programmatically from the microcontroller. This way, all timing signals are easily synchronized.

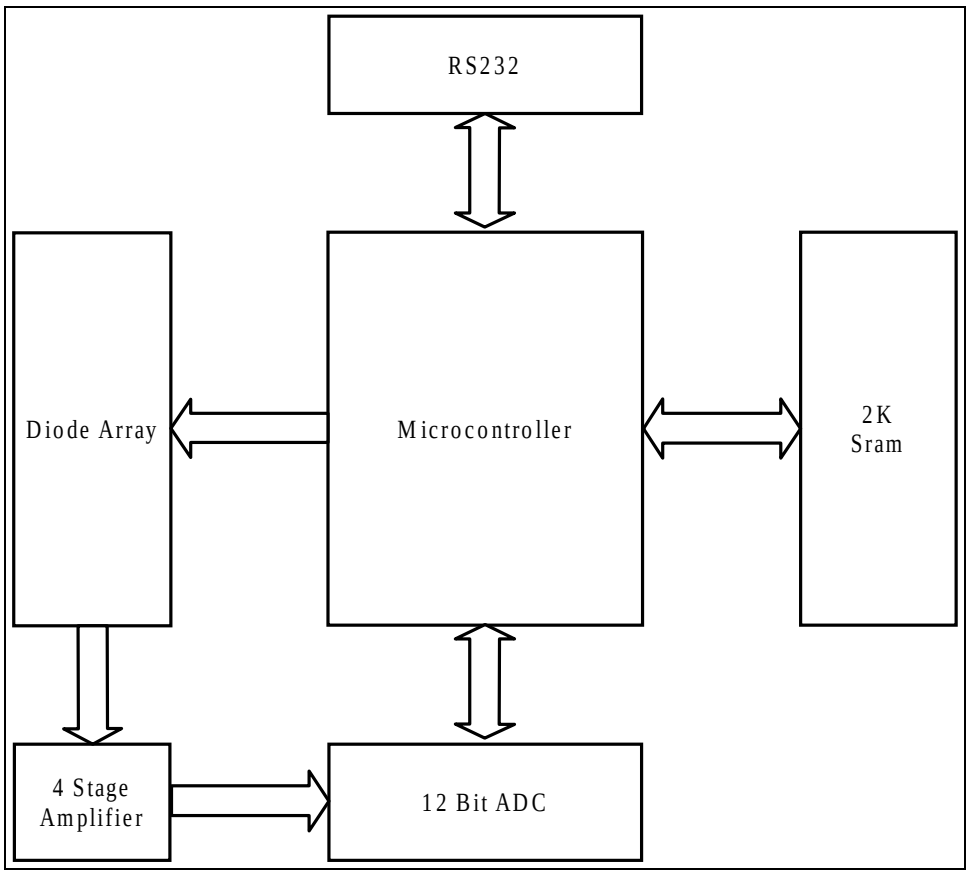

Figure 2

Second, the 16C65A has ample data/IO pins which can each source 20 ma or sink 25 ma. More pins means fewer external parts needed when creating a data/address bus. The large power available per pin means little or no buffering is needed when interfacing to analog components.

The EG&G Reticon 256 channel TB series diode array was chosen as the sensor. An older version has been used with success, so it was a natural progression to choose the state-of-the-art from this company. This sensor has a larger surface area than past versions, so shorter integration times would be the result. Shorter integration times mean faster exposure times leading to sharper ocean color spectra.

The Maxim Max120 12-bit ADC was chosen for its speed. It has a maximum conversion time of 1.6 usecs which would more than keep up with the conversion times needed in this system. It also has a very small form factor and it very stable.

The IDT6166 Static Ram was chosen because it is static ram. No refresh circuitry is needed and it is very fast to access (15ns).

The 74HCT373 octal latch is a standard bus switching device. This version accepts TTL level inputs and outputs CMOS level.

The 74HCT86 XOR gate was needed as a synchronous clock multiplexer needed when driving the diode array. It is of the high speed variety which accepts TTL inputs and outputs CMOS levels.

The Maxim Max313 is a high speed digital switch which acts as a shunt for one of the analog amplifier stages. It too is a fast and reliable part.

Both the DS14C88 and DS14C89 are standard RS232 interface parts. They both drive the IO signals from and to the board to the proper levels. They also provide some degree of noise immunity. Both parts are of the CMOS variety, so they require fewer external parts than their TTL brethren.

The TI084 quad opamp was chosen for speed, low cost, and availability. It is used for analog signal conditioning and amplification. It is a very reliable part that has plenty of bandwidth for this job.

The 12C509 Microchip microcontroller was chosen as a programmable PWM generator needed to drive a servo. It only requires +5 volts and ground to operate, so it was seen as the cleanest solution to a PWM generator. The main microcontroller communicates with this device by simply changing the state on one of its pins. One state causes a PWM to move the servo in one direction. The other state causes a PWM to move the servo in the other direction.

Software

The software for the 16C65A microcontroller can be broken down into 2 main parts. There is a main foreground loop and an interrupt driven background loop. Essentially, the interrupt loop runs with a 10 ms period driven by an internal timer. All data acquisition and control is made in this loop. When a spectrum acquisition is made within this loop, approximately 9 ms of time is used. Since this only occurs at a maximum every 200 ms, plenty of time is left over for the foreground loop.

The foreground loop runs when ever the interrupt loop is not occupied. The foreground loop parses incoming command bytes from the RS232 port. It also checks the queue for data that may have been written by the interrupt loop. When data is found, it is output to the RS232 port. See Figure 3 for a pictorial representation of the 2 loops.

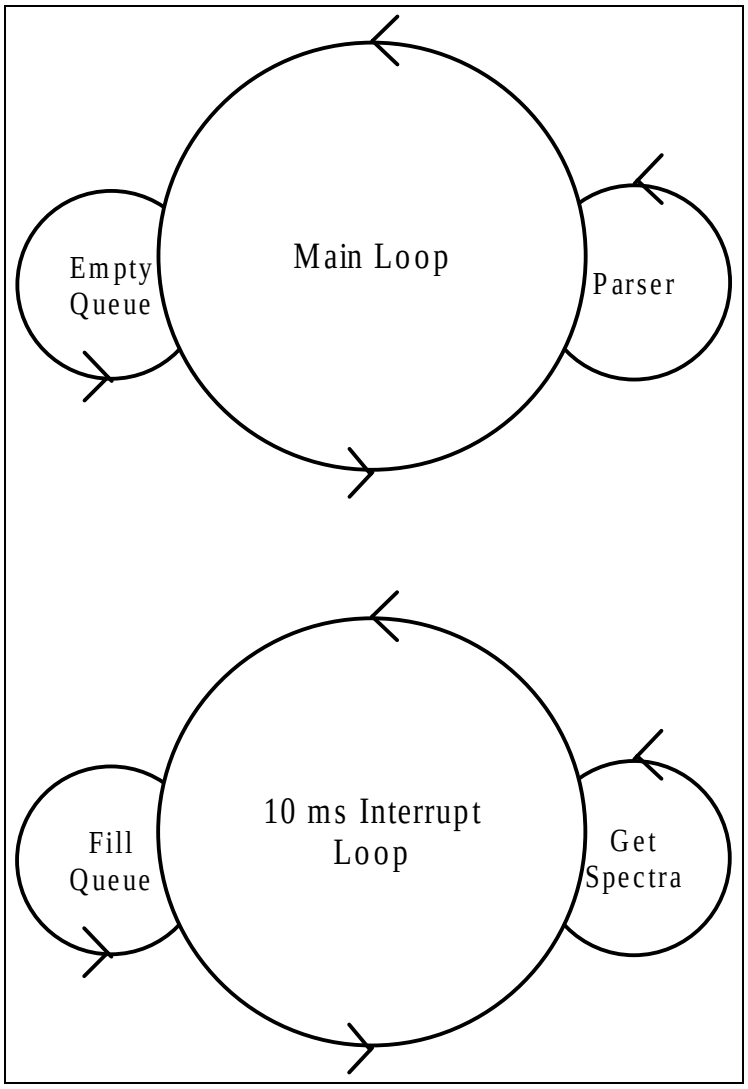

Figure 3

All software is written in Microchip assembly language. Besides normal code comments, I have chosen to include flowcharts of all subroutines. Lets begin with the foreground subroutine called **Start**. This subroutine begins by calling another subroutine called **Initialize**.

Initialize is called only once when the system is turned on. It is here the 16C65A microcontroller has its pins set for the needed input/output directions. The onboard UART is set to 38400 baud, 8 data bits, 1 stop bit, no parity. All external devices are disabled. All variables are cleared. Finally, Timer 1 is set to a 10 ms period which causes an interrupt every cycle. See Figure 4.

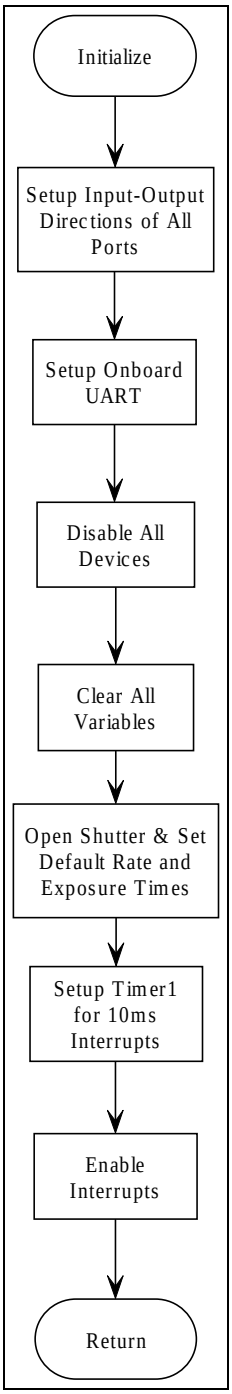

Figure 4

Returning to the **Start** subroutine from **Initialize**, a loop is entered. This loop simply checks 2 conditions. It checks whether there has been a single byte command received on the RS232 port. If a command has been received, the **Parse** subroutine is called. The **Start** loop also checks if the data queue is empty or not. If the queue is not empty, data in the queue is written to the RS232 port. Both of these conditions will be discussed in greater length below. See Figure 5.

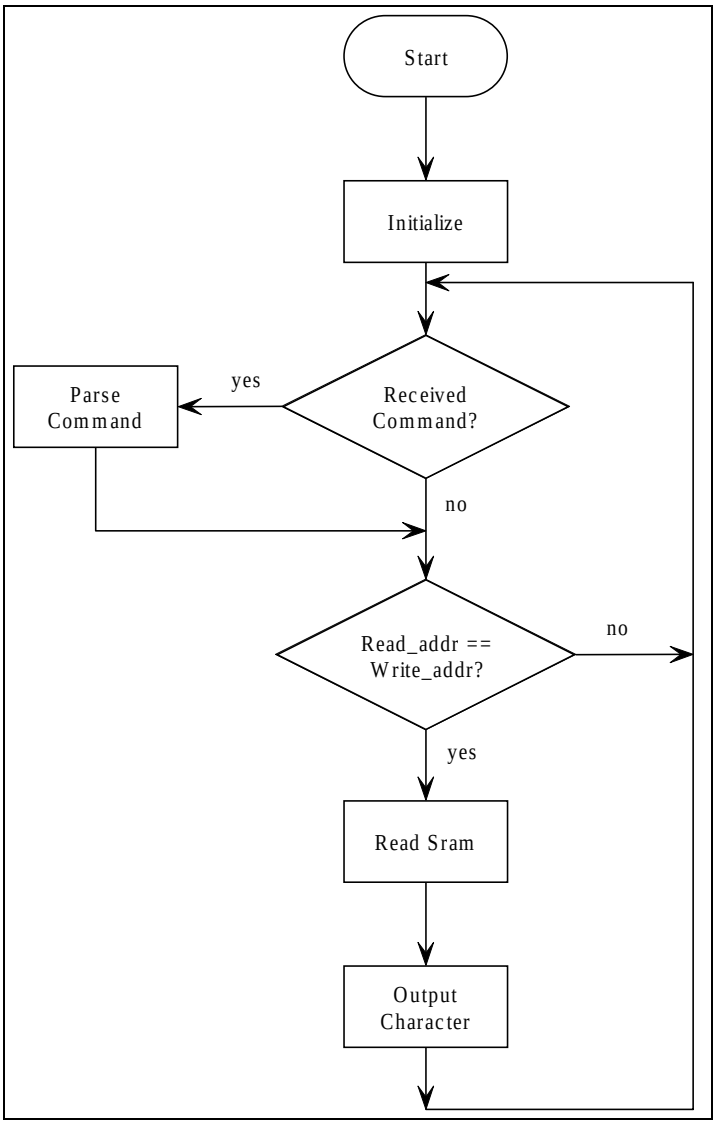

Figure 5

The **Parse** subroutine simply parses any incoming single byte command. The command format is:

Header	**Data**
xxx	xxxxx

The upper 3 bits represent a header or **type** designation. Out of a possible 8 types, 7 are used at the moment. The lower 5 bits represent data. This data has been scaled and has a different meaning for different **types**.

The following table shows the label and binary representation for each command byte.

Command Label	**Header**	**Data**
Set Exposure	000	bbbbb
Set Rate	001	bbbbb
Get One Background	010	xxxxx
Open Shutter	011	xxxx0
Close Shutter	011	xxxx1
Get One Spectrum	100	xxxx0
Get Continuous Spectra	100	xxxx1
Get 24-bit Time	101	xxxxx

x - anything can go here…it is ignored by **Parse**.
b - scaled binary data is required here.
1 - a binary '1' is expected here.
0 - a binary '0' is expected here.

For the **Set Exposure** command, the following table shows the relationship between the 5 data bits and the realized engineering units.

Data Bits	Time (msecs.)	Data Bits	Time (msecs.)
00000	10	10000	170
00001	20	10001	180
00010	30	10010	190
00011	40	10011	200
00100	50	10100	210
00101	60	10101	220
00110	70	10110	230
00111	80	10111	240
01000	90	11000	250
01001	100	11001	260
01010	110	11010	270
01011	120	11011	280
01100	130	11100	290
01101	140	11101	300
01110	150	11110	310
01111	160	11111	320

For the **Set Rate** command, the following table shows the relationship between the 5 data bits and the realized engineering units.

Data Bits	Time (msecs.)	Data Bits	Time (msecs.)
00000	100	10000	1700
00001	200	10001	1800
00010	300	10010	1900
00011	400	10011	2000
00100	500	10100	2100
00101	600	10101	2200
00110	700	10110	2300
00111	800	10111	2400
01000	900	11000	2500
01001	1000	xxxxx	Ignore
01010	1100	xxxxx	Ignore
01011	1200	xxxxx	Ignore
01100	1300	xxxxx	Ignore
01101	1400	xxxxx	Ignore
01110	1500	xxxxx	Ignore
01111	1600	xxxxx	Ignore

The **Parse** subroutine uses a table when comparing the current received command with that of a known command. When a command is found, the offset into that table is used to call the appropriate subroutine via a **Jmptable**. When a command is not found, nothing occurs. See Figure 6.

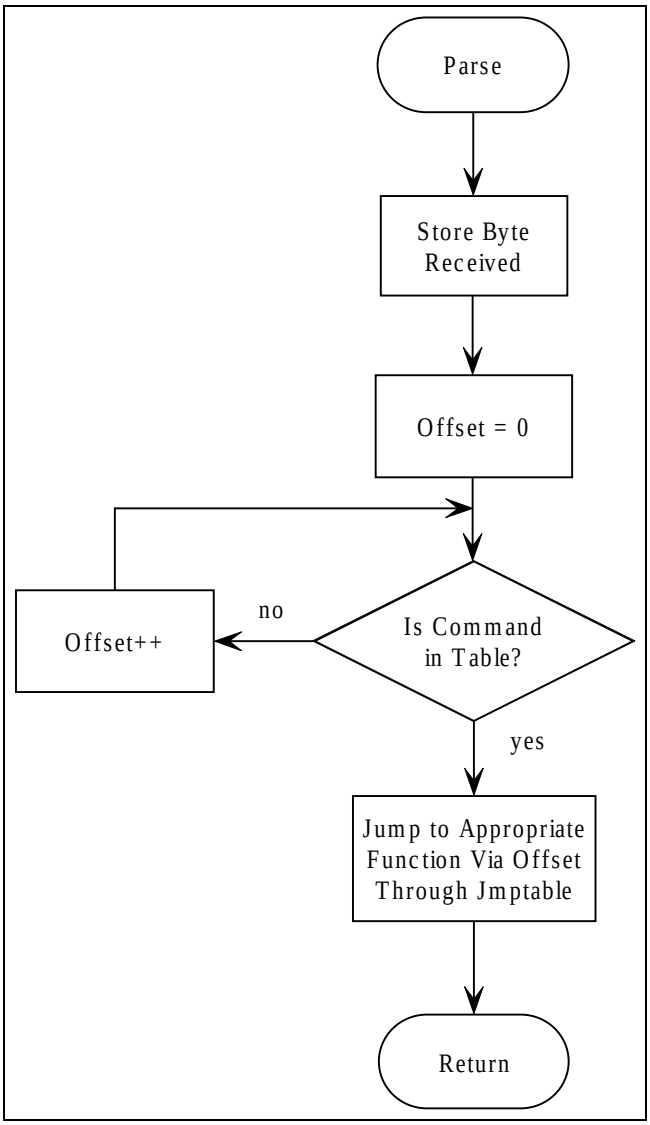

Figure 6

Jmptable is a subroutine that acts as function calling mechanism. At present, there are 6 active functions within **Jmptable**. The **Exposure** function is called when there is change in the exposure time. Exposure times range from 10 ms to 320 ms in increments of 10 ms. The **Rate** function is called when there is a change in the output spectral rate. Rate times range from 100 ms to 2500 ms. The **Background** function simply sets a flag (**back**) that indicates to the **Handler** routine that 1 background spectrum is needed. The **Shutter** routine simply checks the LSB of this command byte. If this bit is set, the shutter is opened. If this bit is zero, the shutter is closed. The **Spectra** routine checks the LSB of this command byte. If this bit is set, the ***spec*** variable is set to 1. This indicates to the **Handler** routine to capture a single spectrum. If this bit is zero, the ***spec*** variable is set to 2. This indicates to the **Handler** routine to capture spectra continuously at the given **Rate** and **Exposure**. The **Readtime** routine sets a variable ***rtime*** to 1. This indicates to the **Handler** routine to output the current 24-bit time. Finally, the **Nextbyte**

command is not implemented. It is meant as an extension to the current command set. If and when it is implemented, it will indicate to the receiver that another command byte follows this one different commands. See Figure 7.

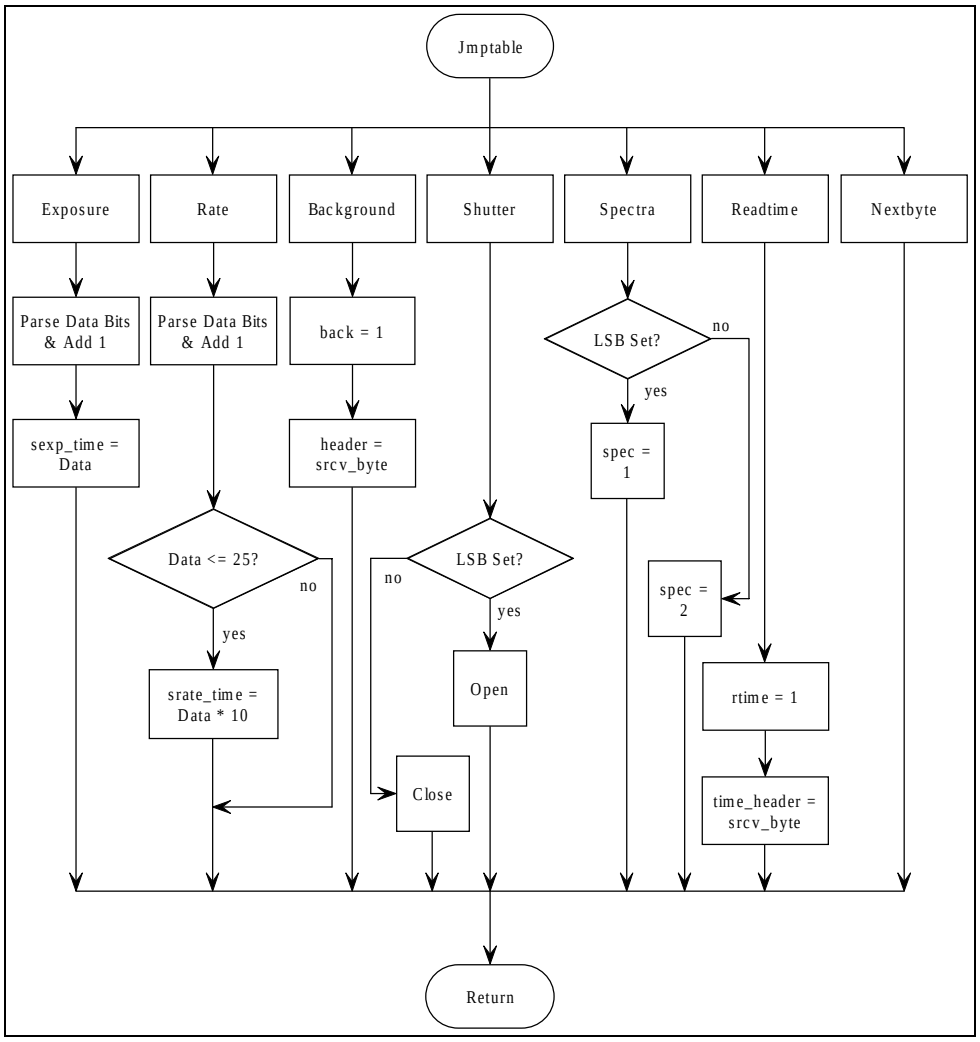

Figure 7

The **Handler** routine is the heart of this data acquisition system. It is here all timing, video clocking and digitization, and data storage is made. This routine is an interrupt handler and has a period of 10 ms. This routine starts by clearing all interrupt flags, incrementing the 24-bit timer variables, and decrementing the rate variable.

This routine basically has 2 parts. The left part of the flowchart only occurs when the exposure flag is set. This flag remains set while the exposure time (***exptime***) counts down. When the exposure time reaches zero, a spectrum acquisition occurs and is stored to Sram.

The right part of the flowchart only occurs when the exposure flag is not set. When the exposure flag is not set, all control variables are examined. These control variables were discussed earlier in **Jmptable**.

Of special note in the **Handler** routine is the call to **FlushCCD**. This routine makes a quick scan of the diode array. This is needed periodically so the charge on the diode array does not continue to increase. Without this command, the bias from the diode array would fluctuate leading to erroneous data. The call to **FlushCCD_ex** is made to flush the diode array at the same exact rate as when the array is digitized and stored in SRAM. This makes the channel to channel scan time equal. See Figure 8.

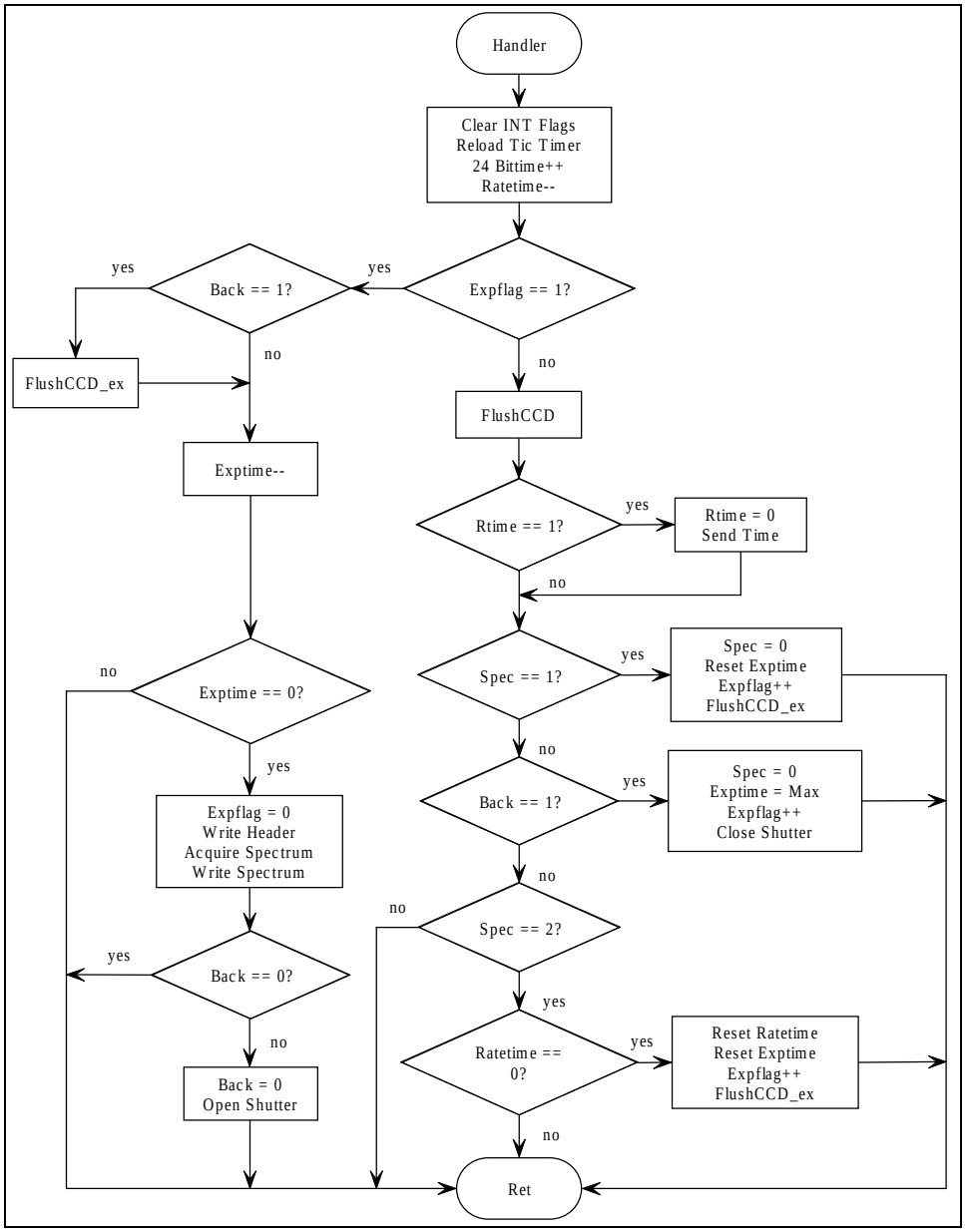

Figure 8

The last 6 routines are timing critical. They must either execute their designated function as quickly as possible or in some finite number of cycles. As mentioned above, **CCDFlush** is a routine that must be called periodically. It quickly reads all 256 channels of the diode array. No data is stored. Reading each channel effectively discharges their respective diode so it is ready to integrate photons again. Flushing the diode is normal done before each exposure begins. It is also done periodically before the next rate command starts up. This way, the diode array is always at some low charged state before it is needed for an actual exposure. See Figure 9.

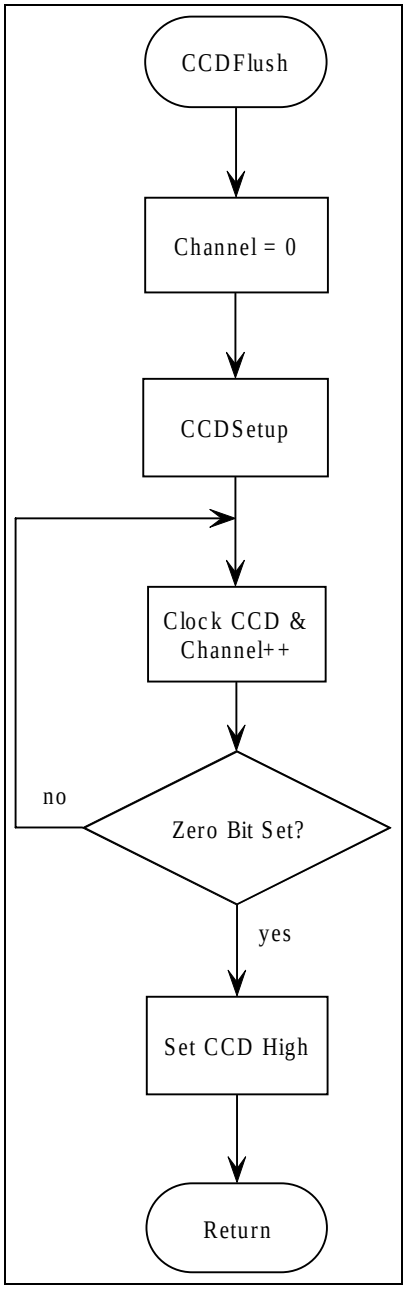

Figure 9

CCDFlush_ex is a routine that is called when a video exposure is started. CCDFlush_ex is essentially an emulation routine that mimics the time necessary to digitize each channel and then store that value to SRAM. This routine is needed so the channel to channel read time is constant when the actual scan is made. Without this routine, later channels are given more integration time then sooner channels…leading to a video exposure that is unbalanced. See Figure 10.

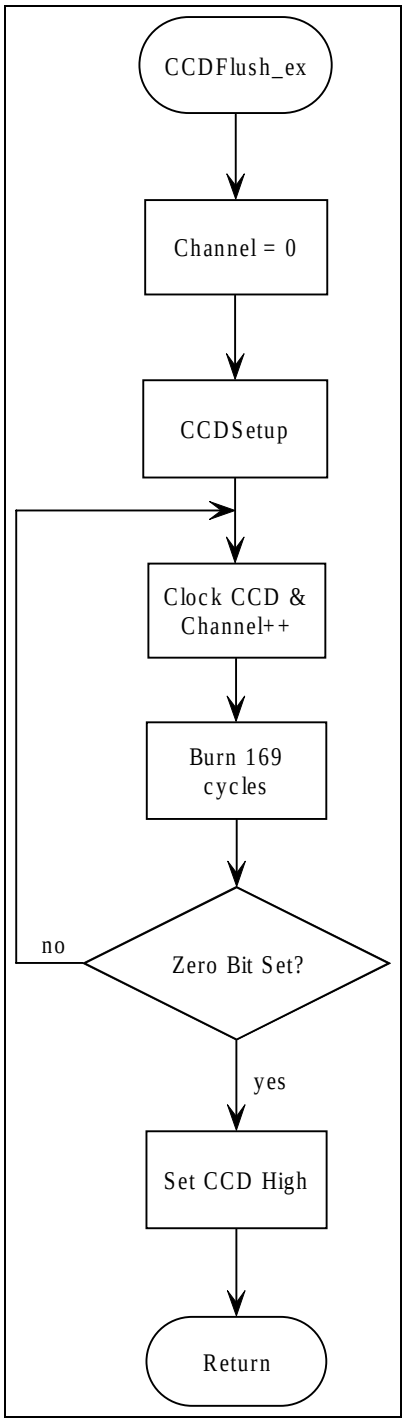

Figure 10

ReadADC is a routine that performs one analog to digital conversion of the video signal coming from the diode array. This routine is hard coded to generate 13 clock cycles which drive the MAX120 12-bit ADC chip. From the Maxim literature, this type of conversion is considered the Slow-Memory Mode. This mode was chosen because it requires the fewest number of control lines to the ADC. Conversion time is very fast.

Of special note in this routine is the re-arrangement of data bits after a conversion. This is done so as to break the 12 returning bits equally into 2-6 bit bytes. The reason for this will be explained in the section on **Output Data Format**. See Figure 11.

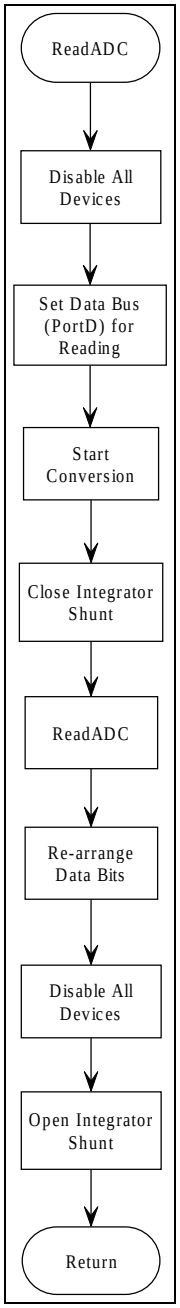

Figure 11

Readsram is a routine that reads one byte from the 2k static ram. Every call to this routine automatically increments a read pointer. This ensures the next call to this routine will fetch the next byte in memory. This routine automatically resets the read pointer to the sram start address when the end of memory is found. See Figure 12.

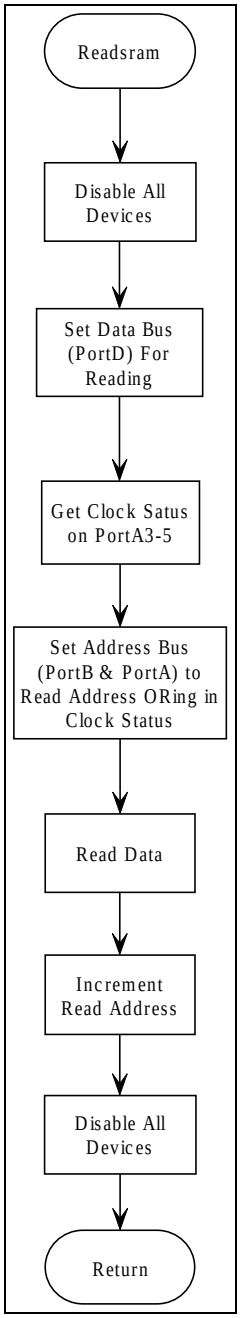

Figure 12

Writesram is a routine that writes one byte to the 2k static ram. Every call to this routine automatically increments a write pointer. This ensures the next call to this routine will write to the next byte in memory. This routine automatically resets the write pointer to the sram start address when the end of memory is found. See Figure 13.

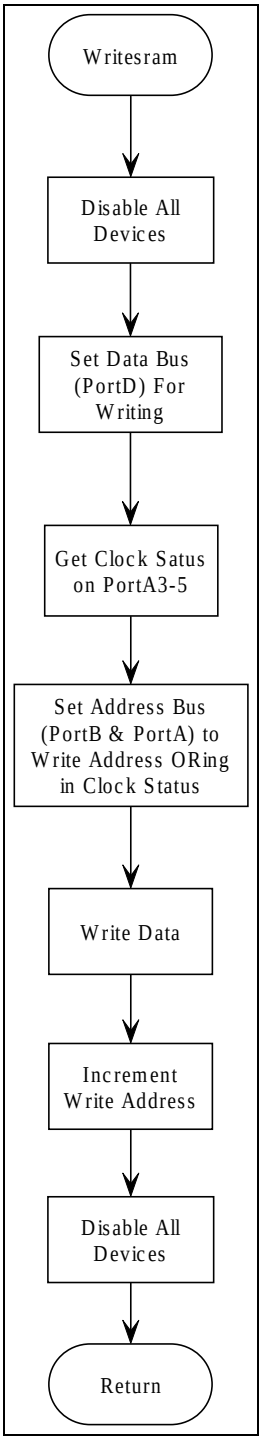

Figure 13

Restore is a routine that restores the current read state to the 8 data I/O lines. This is needed because the read and write lines are multiplexed and are used both in the foreground and background routines. Essentially, this routine allows preemptive multitasking to occur with the data lines. See Figure 14.

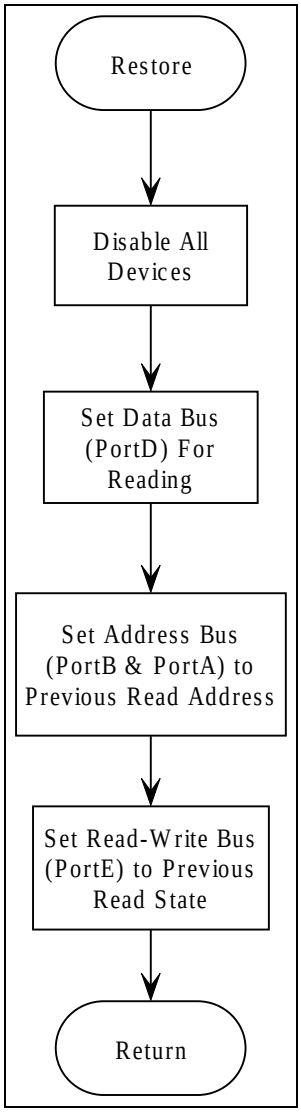

Figure 14

Source Code

```
;-------------------------------------------------------------------------
; Adas18.asm
;           This version attempts to exercise all devices while using
;           the interrupt timer.  A 12-bit AD conversion occurs
;           every 10 msec and 2 bytes are written to the SRAM within
;           the interrupt loop.  The foreground loop 'chases' the
;           write pointer and reads data out of the SRAM.  The
;           foreground loop then sends this data out the serial port.
;           This version adds on to the work done by 'adas11.src' by
;           checking the UART receiver for any data (single byte
;           commands).
;
;           This particular version has been altered to drive the
;           actual CCD device (spectrometer).
;
;           This version (adas17.asm) has been modified to make upper
;           2 bits of data words always '11'.  To do this, I had to
;           shift contents of 'adc_low' and 'adc_high' so each carries
;           6 bits.  This was done as a quick means of data
;           syncronization because command words NEVER have upper 2
;           bits set.
;
;           This version also implements the output of 24 bit time
;           with a 10 msec resolution.  Output is in 4 bytes with
;           6 bits of data each.  Top 2 bits marked 1's to indicate
;           data bytes.  This is our current convention.
;
;           This version creates a new CCD Flush routine called
;           ccdflush_ex for 'exact'.  The purpose of this routine is
;           to flush the CCD at the saime exact rate as when the CCD
;           is actually read and stored into SRAM.  This routine has
;           been written to eliminate the non-uniform bias across all
;           all channels due to the original method of flushing the
;           CCD faster than it is scanned.
;-------------------------------------------------------------------------
; Steven A. Bailey      NASA        4/7/98
;-------------------------------------------------------------------------
      list        p=16c65a, r=dec   ; Define processor and 'dec' numbers
                                    ; as default

      include <p16c65a.inc>         ; Processor specific defines
      include <pxmacs.inc>          ; Parallax 'like' macros

                                    ; These are the configuartion bits
                                    ; found in 'p16c65a.inc'.
      __config    _HS_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF & _BODEN_ON

;-------------------------------------------------------------------------
; Program defines
;-------------------------------------------------------------------------

#define     SPBRG_VAL   7     ; Baud Rate = low
                              ; SPBRG_VAL = (FOSC - BAUDRATE*64)
                              ;             --------------------
                              ;                 (BAUDRATE*64)
                              ;
                              ; For 38.4 Kbaud:
```

```
                                ; SPBRG_VAL = (20,000,000 - 38400*64)
                                ;             ----------------------
                                ;                 (38400*64) = 7

#define     LOW_TIME    0xb9    ; Low byte of startup time
#define     HIGH_TIME   0x3c    ; High byte of startup time
                                ; Since this is a count-up timer
                                ; 10000h - 03cb0h = 0c350h or 50000
                                ; Since a timer tic is 200 ns, then
                                ; 50000 x 200 ns = 0.01 secs between
                                ; interrupts.  I added 9 to calculated
                                ; value because it takes 9 cycles to
                                ; get to handler after interrupt.  This
                                ; was determined using simulator

#define MAX_EXP         200     ; This is time (2000 ms.) needed to
                                ; allow shutter to close to taking 1
                                ; background exposure.

#define CCD_OSTART PORTA,3      ; CCD start (OSTART) pin
#define CCD_OCLOCK PORTA,5      ; CCD clock (OCLOCK) pin
#define CCD_INTEG  PORTC,4      ; CCD integrator reset pin
#define ADC_CS     PORTC,3      ; ADC chip select (CS) pin
#define ADC_INT    PORTC,1      ; ADC INT/BUSY pin
#define ADC_CLKIN  PORTC,0      ; ADC CLKIN pin
#define LATCH_EN   PORTC,5      ; Latch chip enable pin
#define PWM        PORTC,2      ; PWM toggle for P12C509
#define SRAM_CS    PORTE,2      ; SRAM chip select (CS) pin
#define SRAM_RD    PORTE,1      ; SRAM read select (RD) pin
#define SRAM_WR    PORTE,0      ; SRAM write select (WR) pin

;-----------------------------------------------------------------------
; Variables in RAM
;-----------------------------------------------------------------------
; A more consistant way to define RAM is to use the 'org' statement
; followed by variables using the 'res' directive.  However, the RAM
; window of 'MPLAM' does not display variables defined this way...hence
; we are using the 'equ' directive followed by the address in the
; register map they are located.
;-----------------------------------------------------------------------
;           org   0x20        ; Start of RAM in 16c65a
;
;var1       res   1           ; Examples using 'res'
;var2       res   1
;-----------------------------------------------------------------------

rhigh_addr  equ   0x20        ; Read high address byte
rlow_addr   equ   0x21        ; Read low address byte
whigh_addr  equ   0x22        ; Write high address byte
wlow_addr   equ   0x23        ; Write low addequs byte
rdata       equ   0x24        ; Read data byte
wdata       equ   0x25        ; Write data byte
xmt_byte    equ   0x26        ; Byte out to serial port
rcv_byte    equ   0x27        ; Byte in from serial port
srcv_byte   equ   0x28        ; Save byte from serial port
ones        equ   0x29        ; Used for sending BCD
tens        equ   0x2a        ; Used for sending BCD
hunds       equ   0x2b        ; Used for sending BCD
adc_low     equ   0x2c        ; ADC low byte
adc_high    equ   0x2d        ; ADC high byte
rate_time   equ   0x2e        ; 8-bit rate timer
```

```
exp_time    equ   0x2f        ; 8-bit exposure timer
seconds     equ   0x30        ; Our seconds timer
w_copy      equ   0x31        ; Used for stack manipulation
s_copy      equ   0x32
read_state  equ   0x33        ; Current state of read bits
offset      equ   0x34        ; Table offset
element     equ   0x35        ; Returned table element
srdata      equ   0x36        ; Save of rdata
temp        equ   0x37        ; Temp variable
time_low    equ   0x38        ; These 3 for main timer
time_mid    equ   0x39
time_high   equ   0x3a
spec        equ   0x3b        ; Flag for 1spec, specs, or 1back
channel     equ   0x3c        ; Channel number of spectrometer
exp_flag    equ   0x3d        ; Exposure flag
back        equ   0x3e        ; Background flag
srate_time  equ   0x3f        ; Save rate time
sexp_time   equ   0x40        ; Save exposure time
header      equ   0x41        ; Header for return record
clocks      equ   0x42        ; Store state of 2 clock lines
wtemp       equ   0x43        ; Write temp variable
rtemp       equ   0x44        ; Read temp variable
temp_time   equ   0x45        ; This is just for testing
rtime       equ   0x46        ; This is the read time flag
time_header equ   0x47        ; This is storage for readtime header
t_high      equ   0x48        ; Temp storage of high time
t_mid       equ   0x49        ; Temp storage of mid time
t_low       equ   0x4a        ; Temp storage of low time
extra       equ   0x4b        ; Extra byte for output time
burn_cnt    equ   0x4c        ; Burn count variable
;-----------------------------------------------------------------------
; On startup, the PIC looks at address 0 for its first
; instruction. Since the interrupt handler begins at
; address 4, we'll just jump over it to get to the
; startup routine.
;-----------------------------------------------------------------------

        org     0               ; startup vector location
        jmp     start           ; Beginning of main program.

;-----------------------------------------------------------------------

; Next is the interrupt handler, which must begin at
; addequs 4. This handler copies equtoequ w and the status
; register. Because a normal "mov w,fr" alters the z bit of
; the status register, this routine uses "mov w,<>fr," which
; does not. The routine actually swaps the byte twice,
; equulting in the correct value being written to w without
; affecting the z bit.  This routine is setup to be called
; every msec.
;-----------------------------------------------------------------------

      org     4                   ; Interrupt vector location
      jmp   handler               ; Go to timer1 interrupt routine

;-----------------------------------------------------------------------
; Here's the startup routine and the main program loop.
; In the line that initializes "intcon," bit 7 is GIE and bit 5
; is RTIE. Writing 1s to these enables interrupts generally (GIE)
; and the RTCC interrupt specifically (RTIE).
;-----------------------------------------------------------------------
```

```
start
      call  initialize              ; Initialize CPU
mainloop
      snb         PIR1,RCIF         ; Skip if Receiver not ready
      call        parse             ; Parse command from serial port
      cjne_ff     rlow_addr,wlow_addr,output    ; Read SRAM if NOT equal
      cjne_ff     rhigh_addr,whigh_addr,output  ; Read SRAM if NOT equal
      jmp         mainloop

output
      call        readsram
      mov_ff      xmt_byte,rdata    ; Load xmt_byte with rdata
      call        sendchar          ; Send binary data
      jmp         mainloop

;---------------------------------------------------------------------
; Here's our interrupt timer routine.
; In the line that initializes INTCON, bit 7 is GIE and bit 6
; is PEIE.  At the moment, the interrupt handler is being called
; every 10 ms or at a 100 Hz rate.
; From call to return, this function takes 44975 cycles (worst case) or
; 8.99 msecs.  This worst case occurs at most, once every 200 msecs...or
; once every 20 calls to this interrupt handler.
; Average number of cycles used in this function when not writing data
; is 133 cycles or 26.6 usecs.
;---------------------------------------------------------------------

handler
      clrb        STATUS,RP0        ; Switch to Memory Bank 0
      clrb        PIR1,TMR1IF       ; Clear TRM1 interrupt flag

      mov_fw      w_copy,w          ; Make a copy of w.
      mov_ff      s_copy,STATUS     ; Make a copy of status.
      mov_fl      TMR1L,LOW_TIME    ; Load low byte of TMR1
      mov_fl      TMR1H,HIGH_TIME   ; Load high byte of TMR1

      inc         time_low          ; This is our main 24 bit timer
      snz                           ; Resolution is 10 msec giving
      inc         time_mid          ; timer range 2^24 * 10 msec
      snz                           ; or 167772 secs or 46.6 hours
      inc         time_high

      dec         rate_time         ; This timer used for rate.

      cje_fl      exp_flag,1,checkexp  ; If exp_flag == 1, then checkexp

      call        ccdflush          ; If not exposing, must clear CCD
                                    ; Takes approx. 310 usec

      cjne_fl     rtime,1,ckspec    ; Jmp if rtime not set to 1
      clr         rtime             ; Zero rtime flag
      call        sendtime          ; Send out time
ckspec
      cjne_fl     spec,1,ckback     ; Jmp if spec not set to 1
      clr         spec
      inc         exp_flag
      mov_ff      exp_time,sexp_time ; Restore exposure time
      call        ccdflush_ex       ; Flush ccd by reading it…exact time
      jmp         cont
ckback
```

```
      cjne_fl     back,1,ckspecs    ; Jmp if back not set to 1
      call        closeshut         ; Close shutter
      inc         exp_flag
      clr         spec              ; Make sure only 1 spectrum
      mov_fl      exp_time,MAX_EXP  ; Set to MAX exposure time
      jmp         cont
ckspecs
      cjne_fl     spec,2,cont       ; Exit ISR if spec != 2
      cjne_fl     rate_time,0,cont  ; If rate_time not zero, then cont
      inc         exp_flag
      mov_ff      rate_time,srate_time ; Restore rate time
      mov_ff      exp_time,sexp_time   ; Retore exposure time
      call        ccdflush_ex       ; Flush ccd by reading it…exact time
      jmp         cont

checkexp
      cjne_fl     back,1,ckexp      ; Jump if back not set to 1
      call        ccdflush_ex       ; Flush ccd by reading it…exact time

ckexp
      dec         exp_time          ; Decrement exp_time
      jnz         cont              ; Jmp to cont if not zero

      clr         exp_flag          ; Reset exposure flag

      mov_ff      wdata,header      ; Received command is packet header
      call        writesram
      call        ccdsetup          ; Setup CCD for a read
      clrb        CCD_INTEG         ; Open up integrator shunt

adc_loop
      setb        CCD_OCLOCK        ; Clock CCD
      clrb        CCD_OCLOCK

      call        readadc           ; Read ADC
      mov_ff      wdata,adc_low     ; Get lower 8 bits
      call        writesram         ; Write to SRAM
      mov_ff      wdata,adc_high    ; Get upper 4 bits
      call        writesram         ; Write to SRAM
      inc         channel
      jnb         STATUS,Z,adc_loop ; If Z not set, get next channel

      setb        CCD_OCLOCK        ; Get ready for next spectrum
      setb        CCD_INTEG         ; Zero integrator

      cje_fl      back,0,cont       ; If back = 0, then jmp to cont
      clr         back              ; Clear background
      call        openshut          ; Open shutter

cont
      call        restore           ; restore state of possible read
      mov_ff      STATUS,s_copy     ; restore status register
      restw       w_copy            ; restore w without affecting STATUS
                                    ; bits
      reti

;---------------------------------------------------------------------
; Restore current state of possible read
; From call to return, this function takes 27 cycles or 5.4 usecs.
;---------------------------------------------------------------------
```

```
restore
      call        disable           ; Disable all devices

      setb        STATUS,RP0        ; Switch to Memory Bank 1
      mov_fl      TRISD,11111111b   ; Make rd0-7 inputs
      clrb        STATUS,RP0        ; Switch to Memory Bank 0

      mov_ff      PORTB,rlow_addr   ; Store low address


      mov_ff      rtemp,PORTA       ; Need to get state of port
      and_fl      rtemp,00111000b   ; Need bits 3-5
      or_ff       rtemp,rhigh_addr  ; OR in high address
      mov_ff      PORTA,rtemp       ; Restore read state

      mov_ff      PORTE,read_state  ; Done this way for interrupts

      return

;--------------------------------------------------------------------
; sendchar to serial port
; From call to return, this function takes 8 cycles (best case) or
; 1.6 usec.  Worst case dependant on baud rate.
;--------------------------------------------------------------------

sendchar
      sb          PIR1,TXIF         ; Skip if PIR1,TXIF bit set
      jmp         sendchar

      mov_ff      TXREG,xmt_byte    ; Now...transmit xmt_byte

      return                        ; Return to caller

;--------------------------------------------------------------------
; writesram writes to SRAM given:
;
; high_addr -     upper 4 bits
; low_addr  -     lower 8 bits
; data            -     8 bits
;
; This routine has been altered so an equal number of cycles are used
; each time this routine is called...regardless of weather jmps are
; made during the 2 comparisons.
;
; From call to return, this function takes 49 cycles or 9.8 usecs.
;--------------------------------------------------------------------

writesram
      call        disable                 ; Disable all devices

      setb        STATUS,RP0              ; Switch to Memory Bank 1
      mov_fl      TRISD,00000000b         ; Make rd0-7 outputs
      clrb        STATUS,RP0              ; Switch to Memory Bank 0

      mov_ff      PORTB,wlow_addr         ; Store low address

      mov_ff      wtemp,PORTA             ; Need to get state of port
      and_fl      wtemp,00111000b         ; Just need bits 3-5
      or_ff       wtemp,whigh_addr        ; OR in write address

      mov_ff      PORTA,wtemp             ; Store high address
```

```
      nop                                ; Need a 200 ns delay
      mov_ff      PORTD,wdata            ; Store data
      mov_fl      PORTE,00000001b        ; Set WR,CS on & RD off

      inc         wlow_addr
      cjne_fl     wlow_addr,0,not_low
      inc         whigh_addr
      cjne_fl     whigh_addr,8,not_high
      clr         whigh_addr             ; Finished writing 2K SRAM
not_high
      clr         wlow_addr              ; 1 cycle used..this already 0

ret_write
      call        disable                ; Disable all devices
      return                             ; Return to caller

not_low                                  ; Need to burn 6 cycles
      nop
      nop
      nop
      nop
      jmp         ret_write

;-----------------------------------------------------------------------
; readsram reads from SRAM given:
;
; high_addr -     upper 4 bits
; low_addr  -     lower 8 bits
; data            -     8 bits
;
; From call to return, this function takes 53 (worst case) or
; 10.6 usecs.
;-----------------------------------------------------------------------

readsram
      call        disable                ; Disable all devices

      setb        STATUS,RP0             ; Switch to Memory Bank 1
      mov_fl      TRISD,11111111b        ; Make rd0-7 inputs
      clrb        STATUS,RP0             ; Switch to Memory Bank 0

      mov_fl      read_state,00000010b   ; Set RD,CS on & WR off
      mov_ff      PORTB,rlow_addr        ; Store low address

      mov_ff      rtemp,PORTA            ; Need to get state of port
      and_fl      rtemp,00111000b        ; Just need bits 3-5
      or_ff       rtemp,rhigh_addr       ; OR in read address
      mov_ff      PORTA,rtemp            ; Store high address

      mov_ff      PORTE,read_state       ; Done this way for interrupts
      nop                                ; Need a 200 ns delay
      mov_ff      rdata,PORTD            ; Read data from PORTD

      inc         rlow_addr
      cjne_fl     rlow_addr,0,ret_read
      inc         rhigh_addr
      cjne_fl     rhigh_addr,8,ret_read
      clr         rlow_addr              ; Finished reading 2k SRAM
      clr         rhigh_addr

ret_read
```

```
      call        disable                 ; Disable all devices
      mov_fl      read_state,00000100b    ; Set CS off within read_state
      return                              ; Return to caller

;---------------------------------------------------------------------
; Flush CCD.
; From call to return, this function takes 1551 cycles (worst case) or
; 310.2 usecs.
;---------------------------------------------------------------------

ccdflush
      clr         channel
      call        ccdsetup
      setb        CCD_INTEG               ; Zero integrator whole scan

flush_loop
      setb        CCD_OCLOCK
      clrb        CCD_OCLOCK
      inc         channel
      jnb         STATUS,Z,flush_loop     ; If Z not set, get next
                                          ; channel

      setb        CCD_OCLOCK              ; Get ready for next spectrum

      return
;---------------------------------------------------------------------
; Flush CCD Exact.
; From call to return, this function takes 44814 cycles (worst case) or
; 8.96 msecs.  This function emulates the time it takes to scan,
; digitize, and store digitial data in SRAM.
;---------------------------------------------------------------------

ccdflush_ex
      clr         channel
      call        ccdsetup
      setb        CCD_INTEG               ; Zero integrator whole scan

burn_loop                                 ; Each scan takes 35 usecs.
      setb        CCD_OCLOCK
      clrb        CCD_OCLOCK
      call        burn_169                ; Burn 169 cycles
      inc         channel
      jnb         STATUS,Z,burn_loop      ; If Z not set, get next
                                          ; channel

      setb        CCD_OCLOCK              ; Get ready for next spectrum
      setb        CCD_INTEG               ; Zero integrator

      return

;---------------------------------------------------------------------
; Burn 169.
; From call to return, this function takes 169 cycles or 33.8 usecs.
; This function 'burns' off 169 cycles to emulate ADC and SRAM storage
; routines.
;---------------------------------------------------------------------

burn_169
      mov_fl      burn_cnt,40             ; Set burn count variable
                                          ; Causes 160 cycles within
                                          ; loop
```

```
burn_lp
      dec         burn_cnt                ; Takes 1 cycle
      jnb         STATUS,Z,burn_lp        ; Takes 3 cycles when jmp
      nop                                 ; Add extra cycle when 0
                                          ; reached

      nop                                 ; Filler to equal 169 cycles
      nop
      nop

      return

;-----------------------------------------------------------------------
; Setup CCD device for a scan.
; From call to return, this function takes 11 cycles (2.2 usec).
;-----------------------------------------------------------------------

ccdsetup
      clrb        CCD_OCLOCK              ; This sets clock 1 low and
                                          ; sets clock 2 high due to
                                          ; inverter.
      setb        CCD_OSTART              ; Give a start pulse
      clrb        CCD_OSTART

      setb        CCD_OCLOCK              ; Need 2 clock cycles
      clrb        CCD_OCLOCK
      setb        CCD_OCLOCK
      clrb        CCD_OCLOCK

      return

;-----------------------------------------------------------------------
; Read ADC and put low byte in 'adc_low' and high byte in 'adc_high'
; This uses the Slow-Memory Mode.
; From call to return, this function takes 67 cycles or 13.6 usecs.
;-----------------------------------------------------------------------

readadc
      call        disable                 ; Disable all devices

      setb        STATUS,RP0              ; Switch to Memory Bank 1
      mov_fl      TRISD,11111111b         ; Make rd0-7 inputs
      clrb        STATUS,RP0              ; Switch to Memory Bank 0

      clrb        ADC_CS                  ; Start conversion

      setb        ADC_CLKIN               ; Create clock signals.
      clrb        ADC_CLKIN               ; Need 13 cycles for a
      setb        ADC_CLKIN               ; complete ADC
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
```

```
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
      clrb        ADC_CLKIN
      setb        ADC_CLKIN
      clrb        ADC_CLKIN

      setb        CCD_INTEG               ; Close integrator shunt

      mov_ff      adc_low,PORTD           ; Read data from PORTD
      clrb        LATCH_EN                ; Enable latch
      setb        ADC_CS                  ; Disable ADC

      mov_ff      adc_high,PORTD          ; Read data from PORTD

      rl          adc_low                 ; Shift upper bit into C
      rl          adc_high                ; Shift C into lower bit
      rl          adc_low                 ; Shift upper bit into C
      rl          adc_high                ; Shift C into lower bit
      rr          adc_low                 ; Restore position
      rr          adc_low                 ; Restore position

      or_fl       adc_low,11000000b       ; Set upper 2 bits
      or_fl       adc_high,11000000b      ; Set upper 2 bits

      call        disable                 ; Disable all devices

      clrb        CCD_INTEG               ; Open integrator shunt

      return

;---------------------------------------------------------------------
; Disable ADC, latch, and SRAM.
; From call to return, this function takes 7 cycles.
;---------------------------------------------------------------------

disable
      setb        LATCH_EN                ; Disable or flush latch & put
                                          ; in high impedance state
      setb        SRAM_CS                 ; Turn SRAM CS off
      setb        ADC_CS                  ; Turn ADC CS off

      return

;---------------------------------------------------------------------
; Initialize serial port and variables
; From call to return, this function takes 70 cycles or 14 usecs.
;---------------------------------------------------------------------

initialize
      setb        STATUS,RP0              ; Switch to Memory Bank 1

      mov_fl      TRISA,00000000b         ; Make ra0-7 outputs
      mov_fl      TRISB,00000000b         ; Make rb0-7 outputs
      mov_fl      TRISC,10000010b         ; Make rc1 & rc7 pins inputs
      mov_fl      TRISD,00000000b         ; Make rd0-7 outputs
```

```
        mov_fl      TRISE,00000000b         ; Make re0-7 outputs

        mov_fl      SPBRG,SPBRG_VAL         ; Set baud rate generator
        mov_fl      TXSTA,00100000b         ; Trans enable, 8-bits, async,
                                            ; baud low
        mov_fl      PIE1,00000001b          ; Enable TMR1 interrupt on
                                            ; overflow

        clrb        STATUS,RP0              ; Switch to Memory Bank 0

        call        disable                 ; Disable all devices

        clr         rhigh_addr              ; Zero variables
        clr         rlow_addr
        clr         wlow_addr
        clr         whigh_addr
        clr         rdata
        clr         wdata
        clr         adc_low
        clr         adc_high
        clr         seconds
        clr         offset
        clr         time_low
        clr         time_mid
        clr         time_high
        clr         spec
        clr         back
        clr         channel
        clr         exp_flag
        clr         temp_time
        clr         rtime

        clrb        PWM                     ; Clear PWM toggle (open
                                            ; shutter)

        mov_fl      rate_time,100           ; Equivalent rate_time of 1000
                                            ; msec
        mov_fl      srate_time,100
        mov_fl      exp_time,1              ; Equivalent exp_time of 10
                                            ; msec
        mov_fl      sexp_time,1

        mov_fl      read_state,00000100b    ; Make sure CS of read_state
                                            ; is off
        mov_fl      RCSTA,10010000b         ; Serial port enable
        mov_ff      rcv_byte,RCREG          ; Clear out receive buffer

        mov_fl      TMR1L,LOW_TIME          ; Load low byte of TMR1
        mov_fl      TMR1H,HIGH_TIME         ; Load high byte of TMR1
        mov_fl      T1CON,00000001b         ; Turn TMR1 on
        mov_fl      INTCON,11000000b        ; Enable Global and Peripheral
                                            ; Interrupts

        return                              ; Return to caller

;-----------------------------------------------------------------------
; Sendtime command outputs 24 bit time to serial port
; From call to return, this function takes 293 cycles (58.6 usec).
;-----------------------------------------------------------------------

sendtime
```

```
      mov_ff      t_high,time_high  ; Make copy of time_high
      mov_ff      t_mid,time_mid    ; Make copy of time_mid
      mov_ff      t_low,time_low    ; Make copy of time_low

      mov_ff      extra,t_high      ; Times copies upper 6 bits of
                                    ; t_high
      rr          extra             ; Only want 6 bits
      rr          extra
      or_fl       extra,11000000b   ; Mask off upper bits (data bits)

      rl          t_mid             ; Rotate upper 4 bits of t_mid to
                                    ; t_high
      rl          t_high
      rl          t_mid
      rl          t_high
      rl          t_mid
      rl          t_high
      rl          t_mid
      rl          t_high
      or_fl       t_high,11000000b  ; Mask off upper bits (data bits)

      mov_ff      t_mid,time_mid    ; Restore t_mid
      rl          t_low             ; Rotate upper 2 bits of t_low to
                                    ; t_mid
      rl          t_mid
      rl          t_low
      rl          t_mid
      or_fl       t_mid,11000000b   ; Mask off upper bits (data bits)

      mov_ff      t_low,time_low    ; Restore t_low
      or_fl       t_low,11000000b   ; Mask off upper bits (data bits)


      mov_ff      wdata,time_header ; Get ready to write data header
      call        writesram
      mov_ff      wdata,t_low       ; Get ready to write low byte
      call        writesram
      mov_ff      wdata,t_mid       ; Get ready to write mid byte
      call        writesram
      mov_ff      wdata,t_high      ; Get ready to write high byte
      call        writesram
      mov_ff      wdata,extra       ; Get ready to write extra byte
      call        writesram

      return

;---------------------------------------------------------------------
; Parse command from input from serial port
; From call to return, this function takes 215 cycles (worst case) or
; 43 usecs.
;---------------------------------------------------------------------

parse
      mov_ff      rcv_byte,RCREG    ; Read byte from serial port
      mov_ff      srcv_byte,rcv_byte; Save rdata
      and_fl      rcv_byte,11100000b; Only look at upper 3 bits
      clr         offset            ; Start at top of table

ploop
      mov_fl      PCLATH,HIGH table ; Get high address of table
      mov_wf      w,offset          ; Load w with lower 8 bits
```

```
      call        table             ; Get next table entry
      mov_fw      temp,w
      mov_fl      PCLATH,HIGH ploop ; Restore current high address

      and_fl      temp,11100000b    ; Only look at upper 3 bits
      cje_ff      temp,rcv_byte,match     ; Found a match

      inc         offset
      cjb_fl      offset,7,ploop    ; Continue loop if < 7

      return

match
      mov_wf      w,offset
      call        jmptable

      return

;-------------------------------------------------------------------------
; Jmp Table
; From call to return, this function takes 25 cycles (worst case) or
; 5 usecs.
;-------------------------------------------------------------------------

jmptable
      add_fw      PCL,w

      jmp         exposure          ; Jmp to exposure routine
      jmp         rate              ; Jmp to rate routine
      jmp         background        ; Jmp to background routine
      jmp         shutter           ; Jmp to shutter routine
      jmp         spectra           ; Jmp to spectra routine
      jmp         readtime          ; Jmp to readtime routine
      jmp         nextbyte          ; Jmp to nextbyte routine


exposure
      mov_ff      sexp_time,srcv_byte     ; Get saved rcv_byte
      and_fl      sexp_time,00011111b     ; Remove header bits
      inc         sexp_time               ; Remember to add one
      return
rate
      mov_ff      temp,srcv_byte    ; Get saved rcv_byte
      and_fl      temp,00011111b    ; Remove header bits
      inc         temp              ; Remember to add one
      cja_fl      temp,25,rcont     ; Ignore if temp > 25
      clc                           ; Clear carry
      rl          temp              ; Rotate left 1 bit
      mov_ff      srate_time,temp   ; Save rate_time
      rl          srate_time        ; Rotate left 2 bits
      rl          srate_time
      add_ff      srate_time,temp   ; Gives mult. by 10
rcont
      return
background
      mov_fl      back,1            ; Set back flag to 1
      mov_ff      header,srcv_byte  ; Create header
      return
shutter
      mov_ff      rcv_byte,srcv_byte; Restore rcv_byte
```

```
      and_fl      rcv_byte,00000001b; Check LSB for openshut or
closeshut
      jb          STATUS,Z,openshut ; If Z set, must be openshut
closeshut
      setb        PWM               ; Set PWM toggle pin
      return
openshut
      clrb        PWM               ; Clear PWM toggle pin
      return
spectra
      mov_ff      rcv_byte,srcv_byte; Restore rcv_byte
      mov_ff      header,srcv_byte  ; Create header
      and_fl      rcv_byte,00000001b; Check LSB for 1spec or specs
      jb          STATUS,Z,onespec  ; If Z set, must be 1spec
specs
      mov_fl      spec,2            ; Set spec flag to 2
      return
onespec
      mov_fl      spec,1            ; Set spec flag to 1
      return
readtime
      mov_fl      rtime,1           ; Set rtime flag to 1
      mov_ff      time_header,srcv_byte   ; Save current time header
      return
nextbyte
      return

;-----------------------------------------------------------------------
; Table lookup
; From call to return, this function takes 6 cycles or 1.2 usecs.
;-----------------------------------------------------------------------

      org   0x800

table
      add_fw      PCL,w

      retw        00000000b         ; Exposure command
      retw        00100000b         ; Rate command
      retw        01000000b         ; Take 1 background spectrum
      retw        01100000b         ; Shutter command
      retw        10000000b         ; Spectra command
      retw        10100000b         ; Read time
      retw        11000000b         ; Next byte command

        end
```

```
;-------------------------------------------------------------------------
; Pwm1.asm  This program accepts input control from a PIC16C65A
;           and generates the appropriate PWM output on a 12C509.
;           At present, only 2 PWM signals are produced.  These
;           2 signals cause the servo to go from 1 side of its
;           rotation path to another.  The angle covered is
;           approximately 110 deg.
;
;           For Futaba servo control, the PWM frequency determined
;           to work best is 50 Hz (20 msec period).
;
;           The shortpulse signal is ON 200 usec and OFF 19800 usec.
;           This is a duty cycle of 1%
;
;           The longpulse signal is ON 1500 usec and OFF 18500 usec.
;           This is a duty cycle of 8%
;-------------------------------------------------------------------------
; Steven A. Bailey      NASA        3/19/98
;-------------------------------------------------------------------------

      list              p=12c509, r=dec   ; Define processor and 'dec'
                                          ; numbers as default

      include <p12c509.inc>               ; Processor specific defines
      include <pxmacs5x.inc>              ; Parallax 'like' macros

                                          ; These are the configuration
                                          ; bits found in 'p12c509.inc'.
      __config    _IntRC_OSC & _WDT_OFF & _MCLRE_OFF & _CP_OFF

;-------------------------------------------------------------------------
; Program defines
;-------------------------------------------------------------------------

#define PWM_INPUT       GPIO,3      ; PWM control from p16c65a
#define PWM_OUTPUT      GPIO,1      ; PWM output from p12c5xx

;-------------------------------------------------------------------------
; Variables in RAM
;-------------------------------------------------------------------------

pulse_low         equ   0x07        ; Pulse low width
pulse_high        equ   0x08        ; Pulse high width
cycle_low         equ   0x09        ; Remaining cycle low width
cycle_high        equ   0x0a        ; Remaining cycle high width

;-------------------------------------------------------------------------
; On startup, the PIC12C509 looks at address 0 for its first
; instruction. No interrupts to worry about, because they don't exist
; on this chip.
;-------------------------------------------------------------------------

      org         0                 ; startup vector location

      clrb        STATUS,PA0        ; Make sure using Memory Bank 0
      tris_l      00001000b         ; Make gp3 an input & gp5,4,2,1,0
                                    ; outputs
      clr         GPIO

mainloop
```

```
      jb          PWM_INPUT,longpulse ; If pin level high, then do
longpulse
      setb        PWM_OUTPUT        ; else do shortpulse
      call        usec200           ; Pulse on 200 usec
      clrb        PWM_OUTPUT

      call        usec1000          ; Pulse off total of 19800 usecs
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec200
      call        usec200
      call        usec200
      call        usec200
      jmp         mainloop

longpulse
      setb        PWM_OUTPUT
      call        usec1000          ; Pulse on 1500 usec
      call        usec200
      call        usec200
      call        usec100
      clrb        PWM_OUTPUT

      call        usec1000          ; Pulse off total of 18500 usecs
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec1000
      call        usec200
      call        usec200
      call        usec100
      jmp         mainloop
```

```
;-------------------------------------------------------------------------
; This routines burns off time...remember, only a 2 level stack with
; the PIC12C509.
;-------------------------------------------------------------------------

usec1000                        ; Burn off 1000 usec
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10

      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10

      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10

      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10

      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
```

```
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10

        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10

        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10

        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10

        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec10
        call        usec6
        return

usec200                         ; Burn off 200 usec
        call        usec10
        call        usec10
        call        usec10
```

```
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10

      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec6
      return

usec100                       ; Burn off 100 usec
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec10
      call        usec6
      return

usec10                        ; Burn off 10 usec.
      nop                     ; Note...it takes 4 usec to call & return
      nop
      nop
      nop
      nop
      nop
      return

usec6                         ; Burn off 6 usec.
      nop                     ; Note...it takes 4 usec to call & return
      nop
      return

      end
```

Schematics

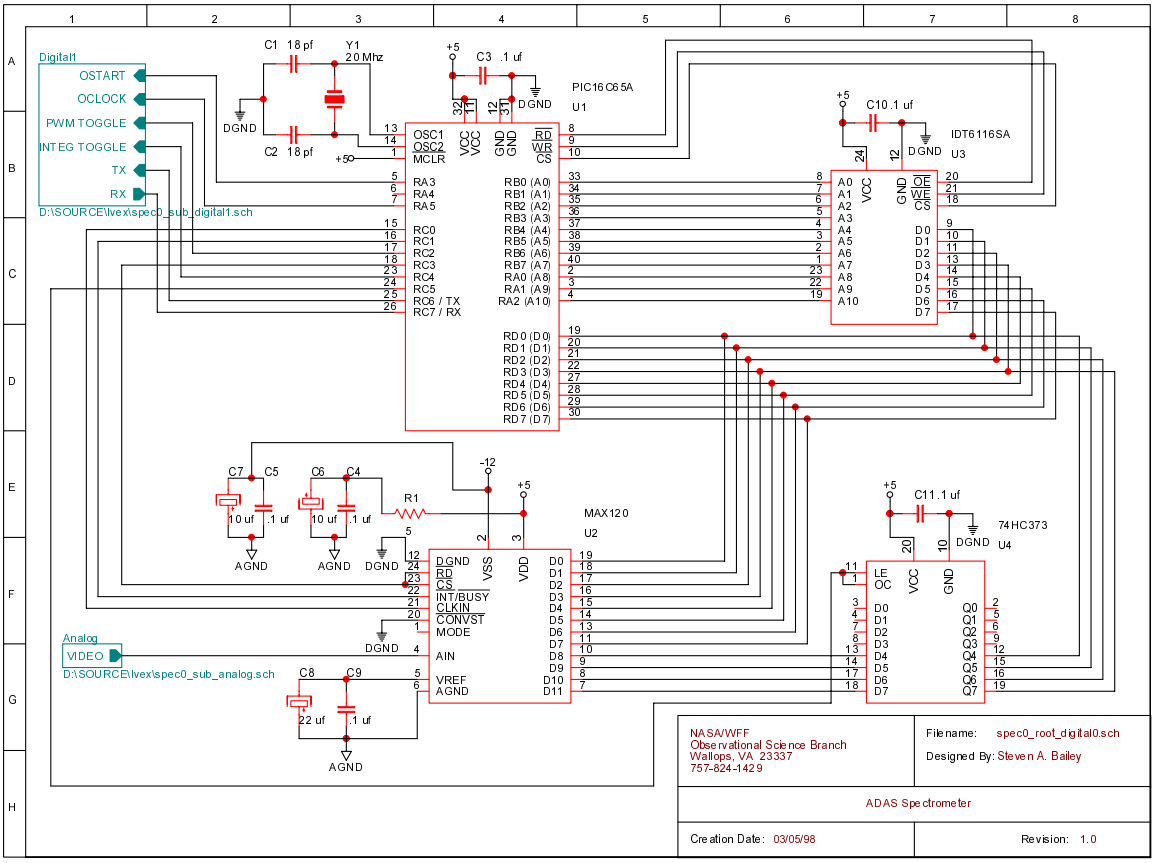

Figure 15

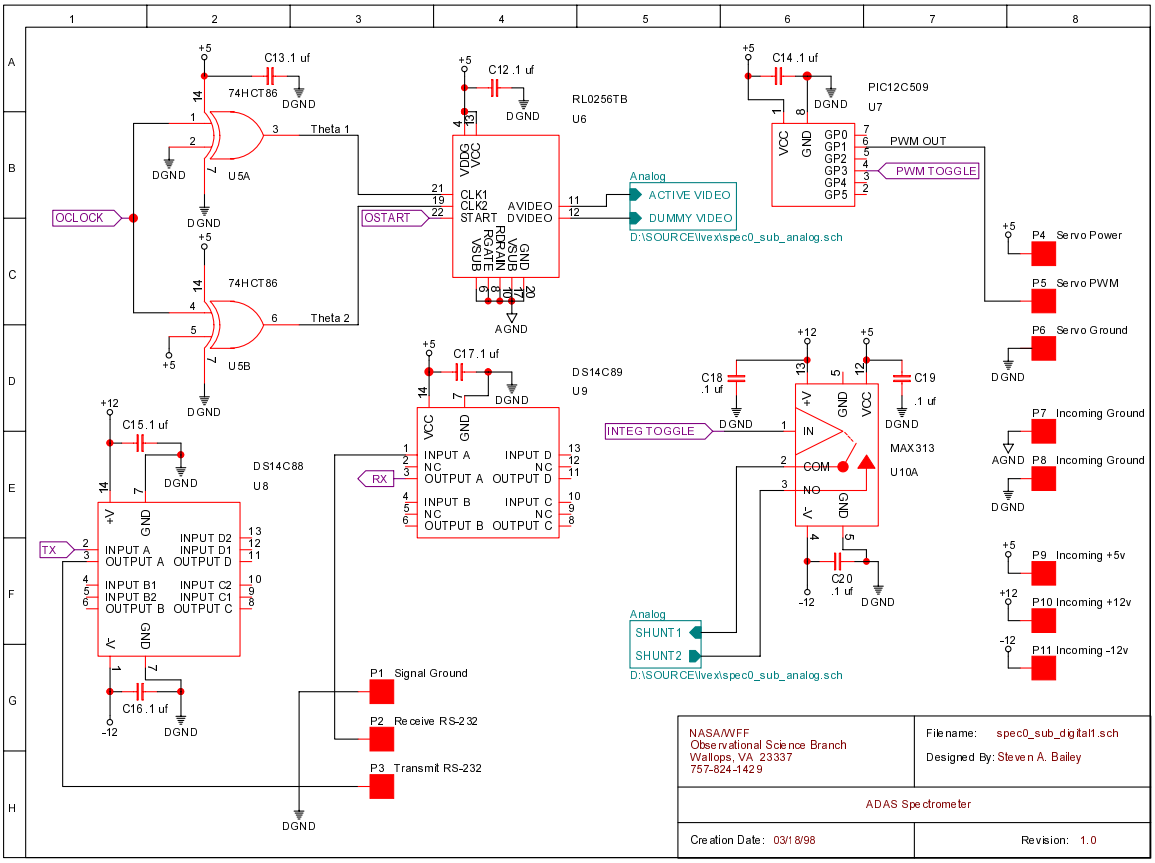
C13.1 uf
74HCT86
U5A
Theta 1
OCLOCK
74HCT86
U5B
Theta 2
C12.1 uf
RL0256TB
U6
OSTART
Analog
ACTIVE VIDEO
DUMMY VIDEO
D:\SOURCE\lvex\spec0_sub_analog.sch
C14.1 uf
PIC12C509
U7
PWM OUT
PWM TOGGLE
P4 Servo Power
P5 Servo PWM
P6 Servo Ground
C17.1 uf
DS14C89
U9
C15.1 uf
DS14C88
U8
TX
RX
C16.1 uf
P1 Signal Ground
P2 Receive RS-232
P3 Transmit RS-232
C18 .1 uf
C19 .1 uf
C20 .1 uf
INTEG TOGGLE
MAX313
U10A
Analog
SHUNT1
SHUNT2
D:\SOURCE\lvex\spec0_sub_analog.sch
P7 Incoming Ground
P8 Incoming Ground
P9 Incoming +5v
P10 Incoming +12v
P11 Incoming -12v
NASA/WFF
Observational Science Branch
Wallops, VA 23337
757-824-1429
Filename: spec0_sub_digital1.sch
Designed By: Steven A. Bailey
ADAS Spectrometer
Creation Date: 03/18/98
Revision: 1.0

Figure 16

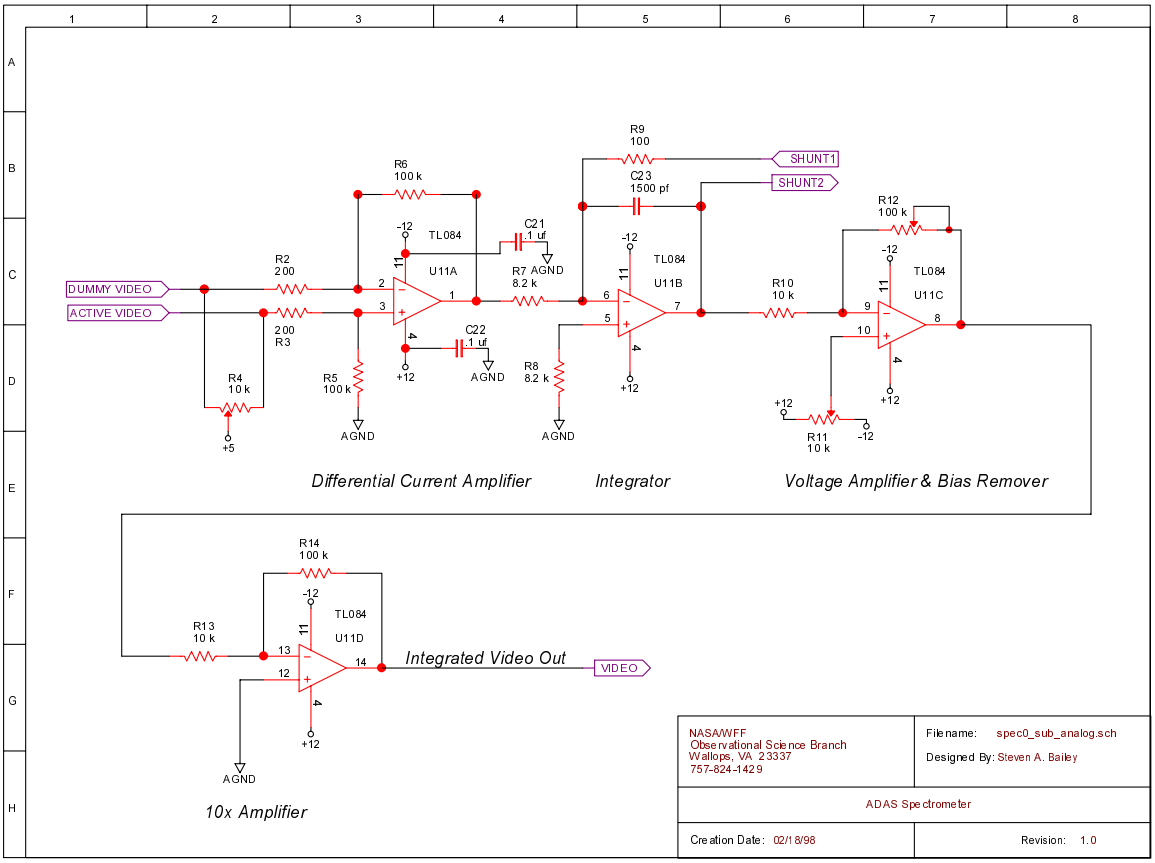
DUMMY VIDEO
ACTIVE VIDEO
R2 200
R3 200
R4 10 k
R5 100 k
R6 100 k
R7 AGND 8.2 k
R8 8.2 k
R9 100
R10 10 k
R11 10 k
R12 100 k
C21 .1 uf
C22 .1 uf
C23 1500 pf
SHUNT1
SHUNT2
TL084
U11A
U11B
U11C
U11D
-12
+12
+5
AGND
Differential Current Amplifier
Integrator
Voltage Amplifier & Bias Remover
R13 10 k
R14 100 k
Integrated Video Out
VIDEO
10x Amplifier
NASA/WFF
Observational Science Branch
Wallops, VA 23337
757-824-1429
Filename: spec0_sub_analog.sch
Designed By: Steven A. Bailey
ADAS Spectrometer
Creation Date: 02/18/98
Revision: 1.0

Figure 17

Analog Section

The following is a scope snapshot of two signals from the analog portion of our circuit. The top trace (signal 1) is our video start pulse which is found on Pin 5 of U1 or Pin 22 of U6. It is approximately 200 ns in width and occurs once at the beginning of video acquisition.

The bottom trace (signal 2) is channel 1 of our 256 channel video signal. It occurs approximately 1.5 usec after our start signal. Signal 2 can be found on Pin 1 of U11. This is the output from our first analog stage. This stage is a differential current amplifier. Take notice that the video signal (signal 2) is approximately 2.6 usec in width.

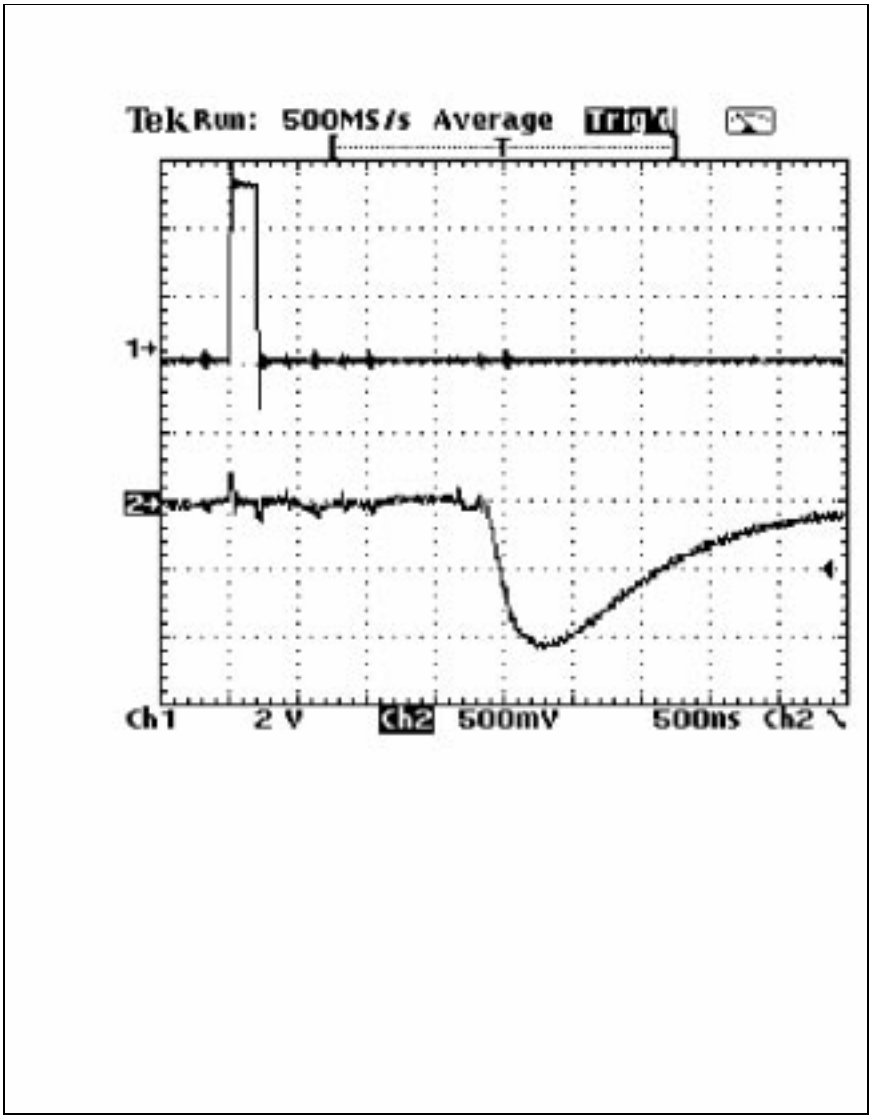

Figure 18

The following is a scope snapshot which shows the relationship between 2 adjacent video signals. Again, signal 1 is our video start pulse. Signal 2 is our video signal spaced to show 2 successive video channels. Channel spacing is approximately 35 usec. This means a full acquisition of 256 channels should take 35 x 256 = 8.96 msec. Take note that is acquisition speed is based on scanning each channel, performing a 12 bit ADC, and saving this digital value to SRAM.

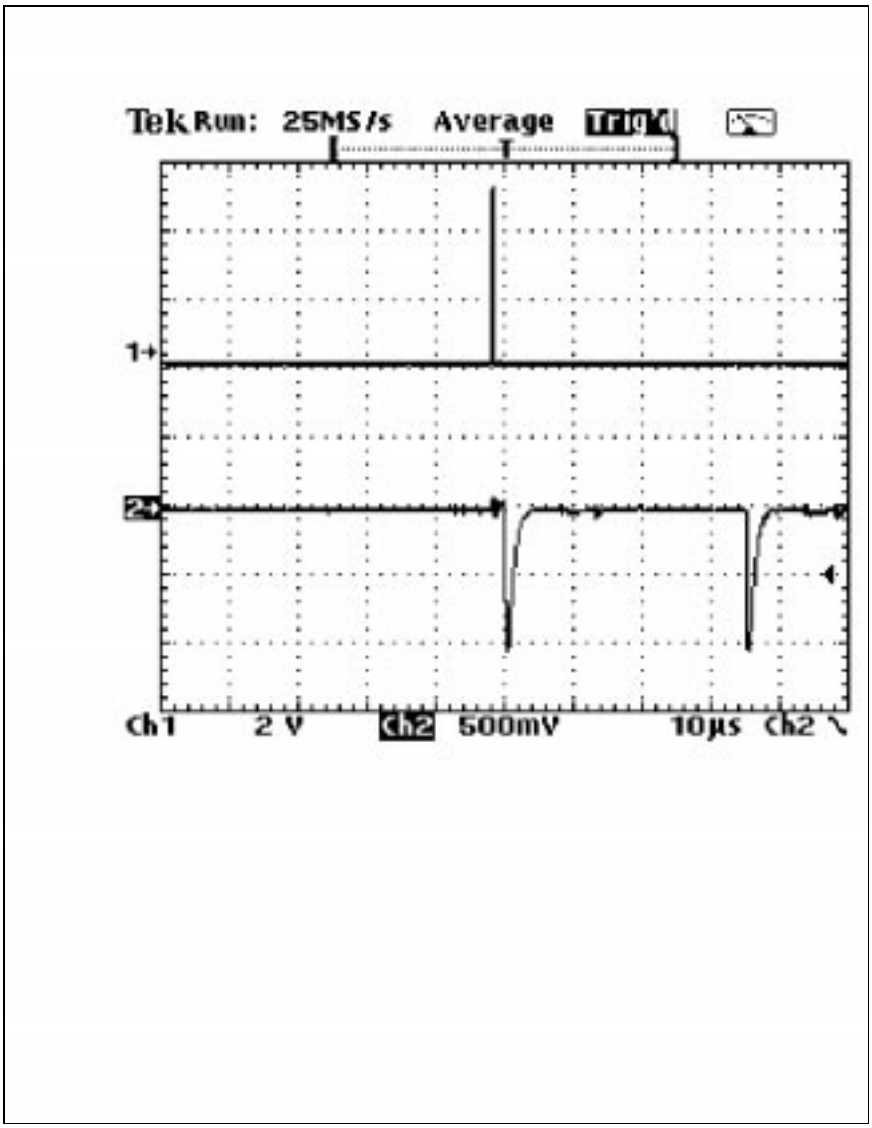

Figure 19

The following is a scope snapshot which shows the relationship between the raw video signal and the integrated signal. Signal 2 is our video signal coming off of Pin 1 of U11. Signal 1 is our integrated signal from Pin 7 of U11. This is our second stage, so the signal is low level at 50 mv/div. The 3 noise spikes surrounding the signal are caused by the integrating shunt caused by U10. This is normal and does not affect our A/D sample.

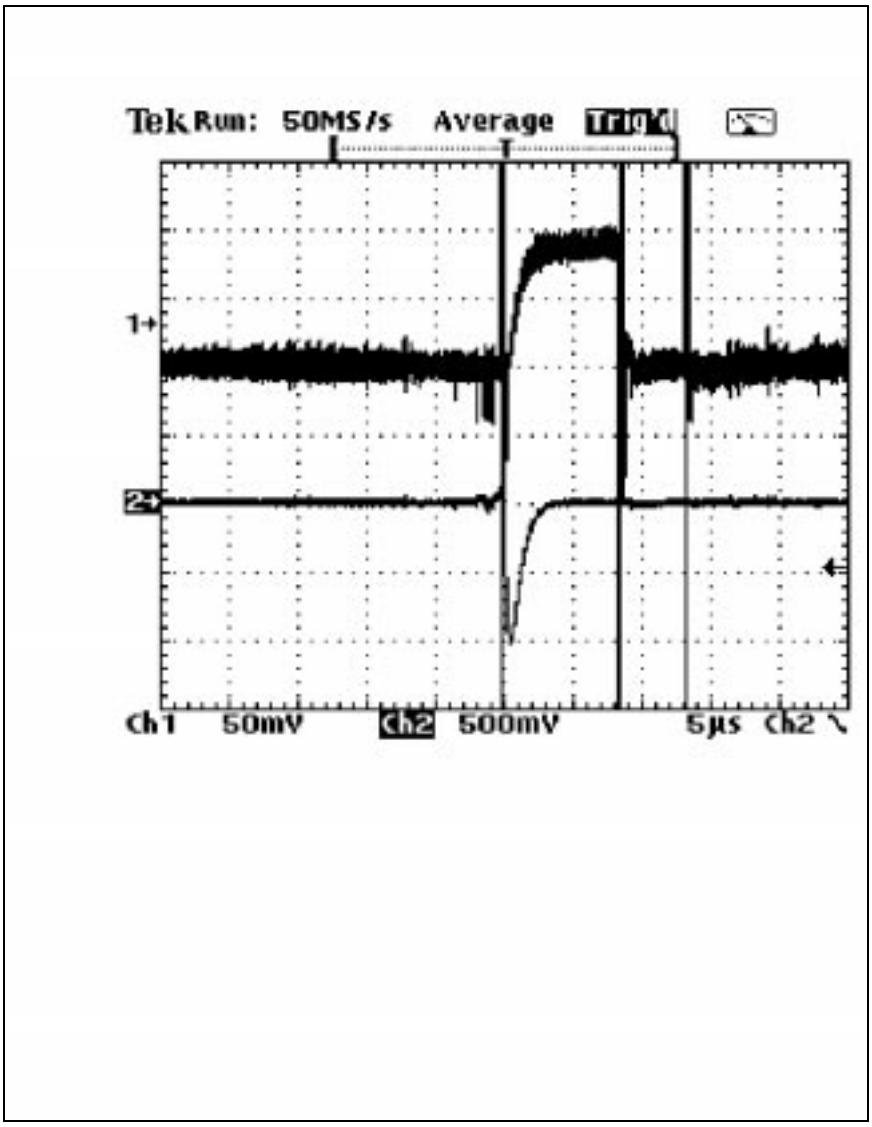

Figure 20

The following is a scope snapshot which shows the relationship between the raw video signal and the final integrated signal. Signal 2 is our video signal coming off of Pin 1 of U11. Signal 1 is our integrated signal from Pin 14 of U11. This is our fourth stage, so the signal is high level at 5 v/div.

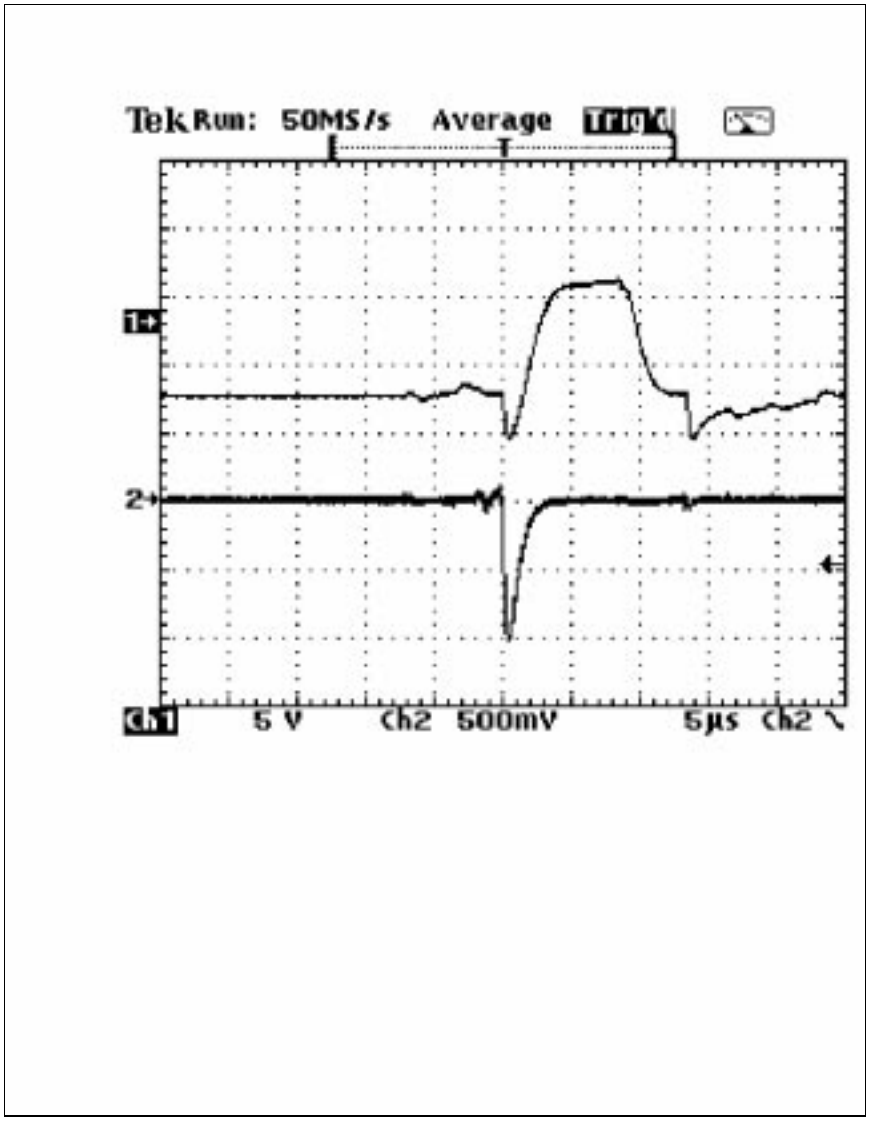

Figure 21

The following is a scope snapshot which shows the relationship between the integrated video signal and the integration shunt. Signal 1 is our integrated signal from Pin 14 of U11. Signal 2 is the TTL shunt signal arriving on Pin 1 of U10. When this signal is high, the shunt is ON and the current integration signal across C23 is grounded.

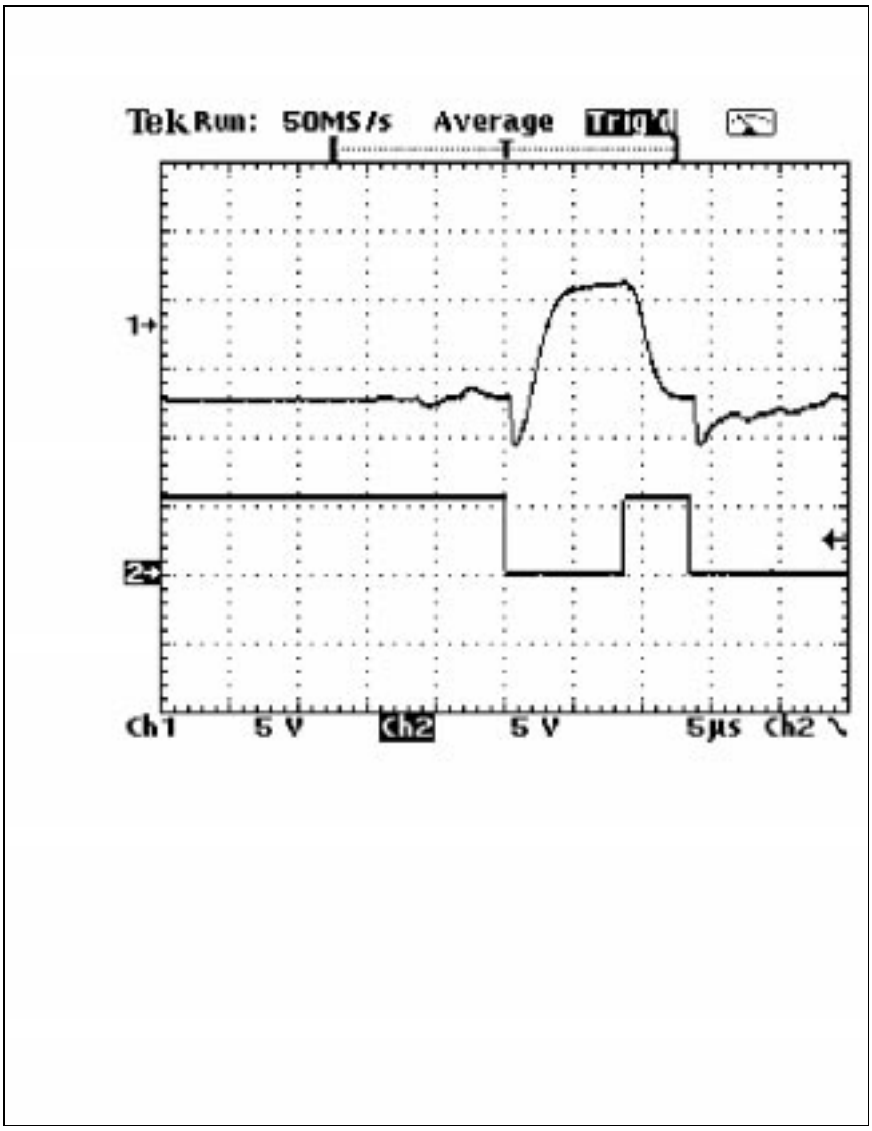

Figure 22

The following is a scope snapshot which shows the relationship between the integrated video signal and the integration shunt signal. Signal 1 is our integrated signal from Pin 14 of U11. Signal 2 is the TTL shunt signal arriving on Pin 1 of U10. This figure differs from the previous figure in that the entire video frame is captured here. As expected, our video frame is approximately 9 ms in width. Also, since our light source for this test was an LED shining equally across all channels…the signal is equal for all channels.

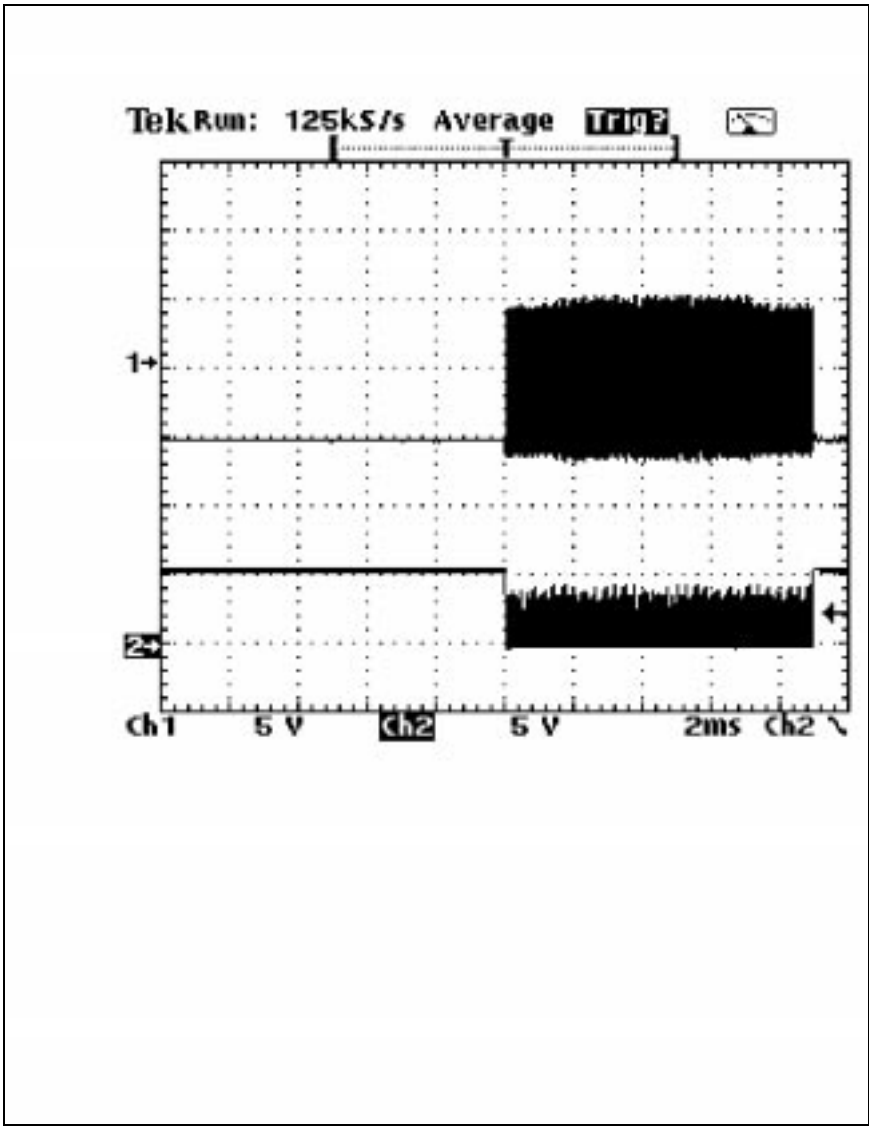

Figure 23

The following figure is the digital display of the video signal from the previous figure. This signal was digitized with the said 12-bit A/D, buffered, and sent to a host PC via RS-232. There is an apparent roll-off at the edges which is probably due to reflections at each end of the diode array. This aberration should vanish when a diffuser is placed over the diode array.

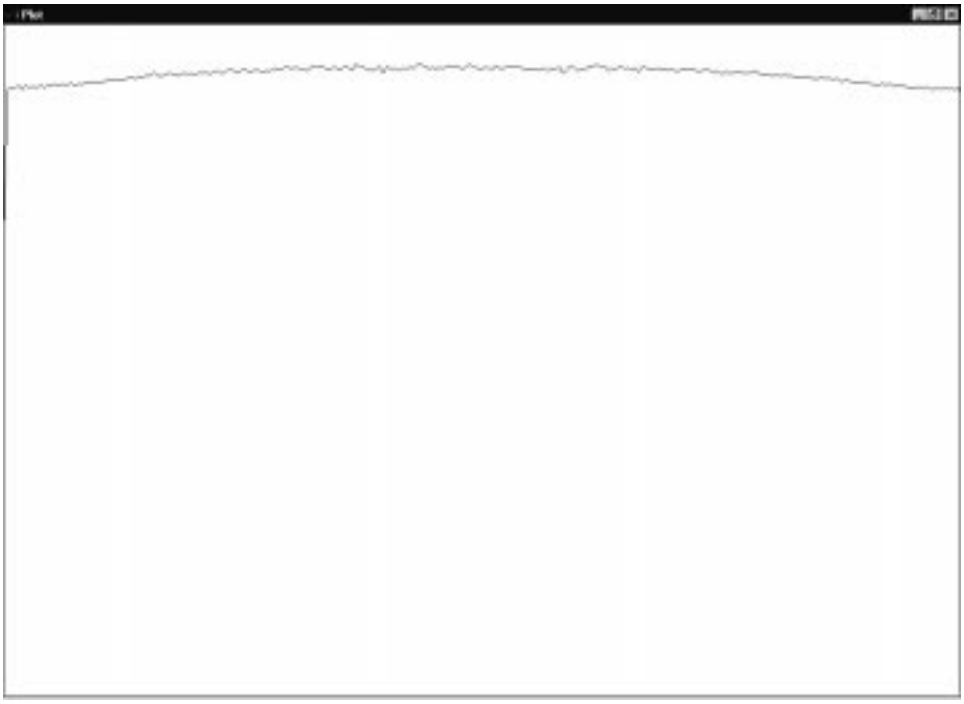

Figure 24

The following is a scope snapshot which shows the relationship between the integrated video signal and the integration shunt signal. Signal 1 is our integrated signal from Pin 14 of U11. Signal 2 is the TTL shunt signal arriving on Pin 1 of U10. This figure differs from the previous figure in that only ~10 channels of the entire video frame are captured.

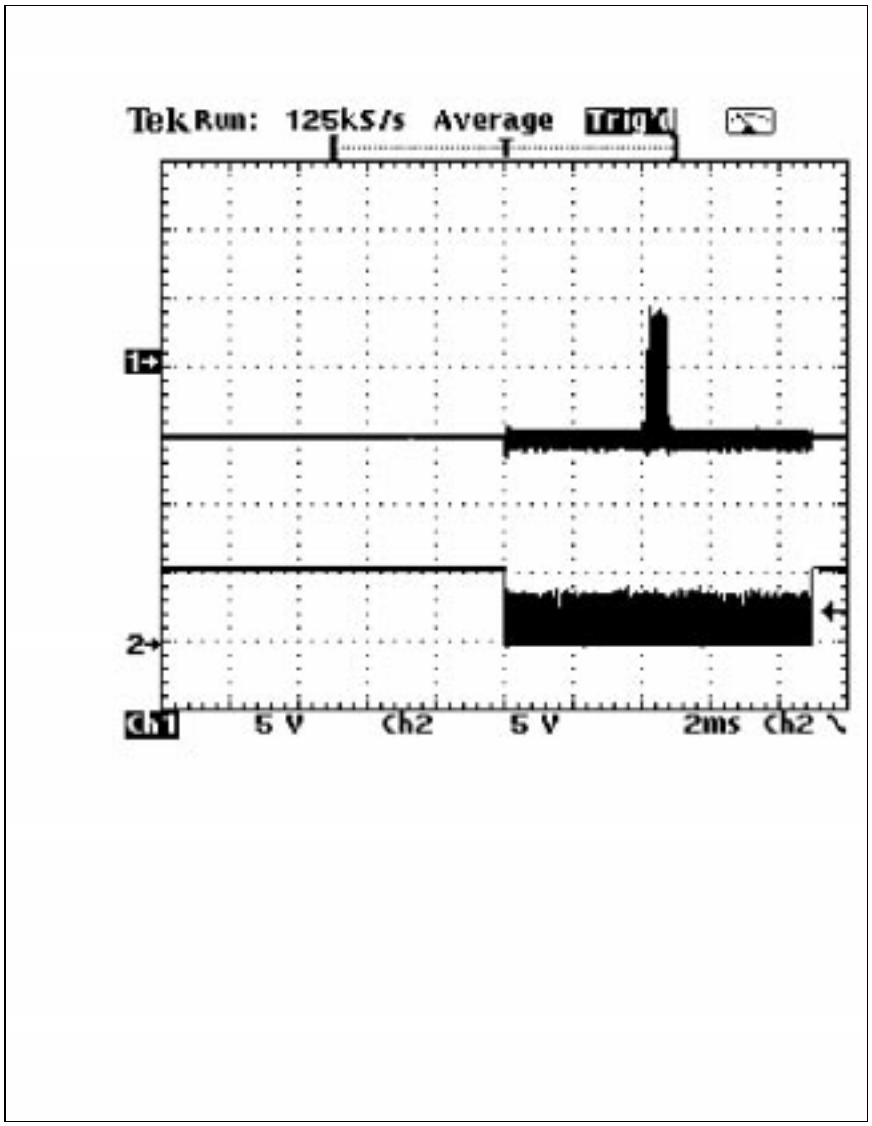

Figure 25

The following figure is the digital display of the video signal from the previous figure. This signal was digitized with the said 12-bit A/D, buffered, and sent to a host PC via RS-232.

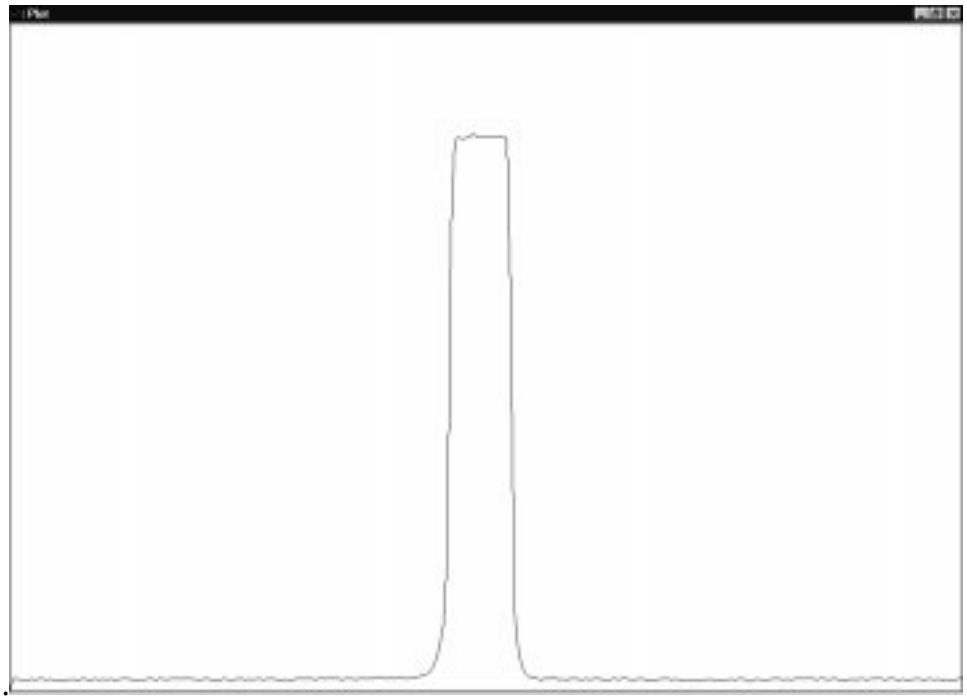

Figure 26

The following is a scope snapshot which shows the relationship between the integrated video signal and the integration shunt signal. Signal 1 is our integrated signal from Pin 14 of U11. Signal 2 is the TTL shunt signal arriving on Pin 1 of U10. This figure differs from the previous figure in that only ~2 channels of the entire video frame are captured.

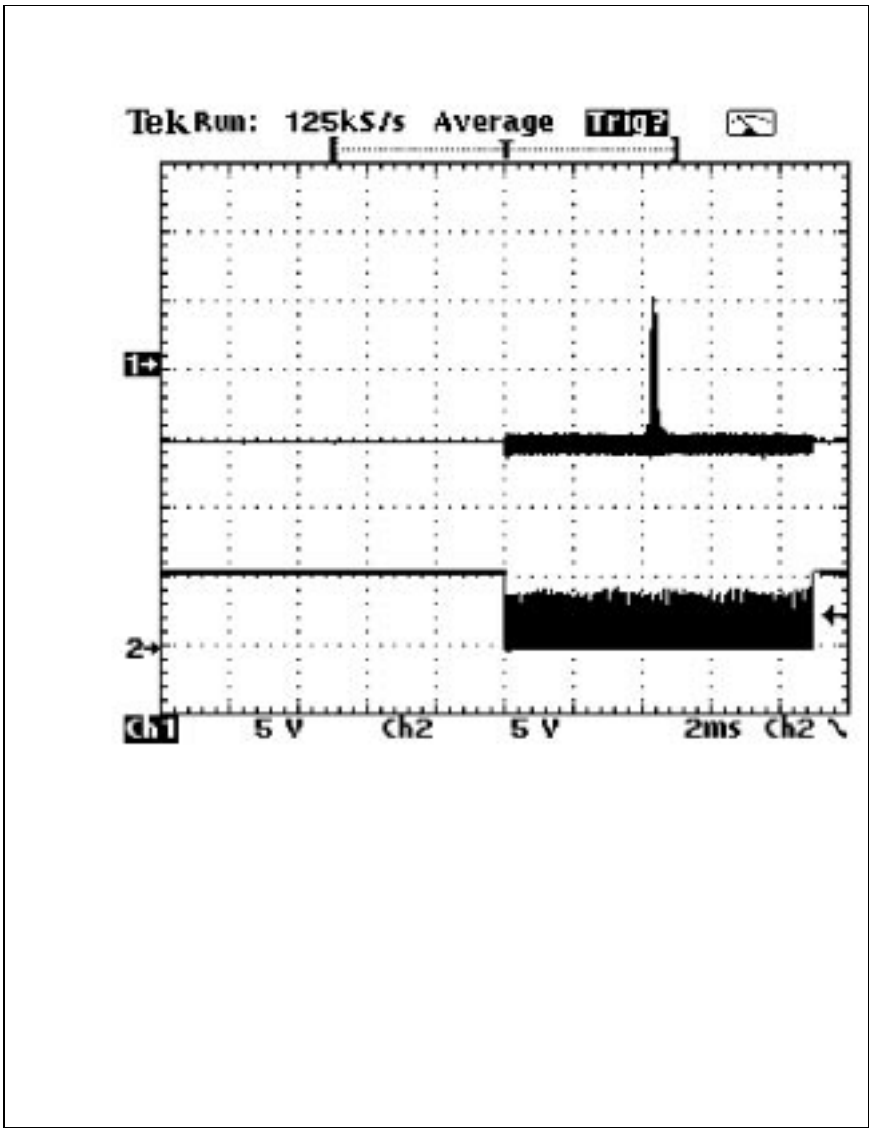

Figure 27

The following figure is the digital display of the video signal from the previous figure. This signal was digitized with the said 12-bit A/D, buffered, and sent to a host PC via RS-232.

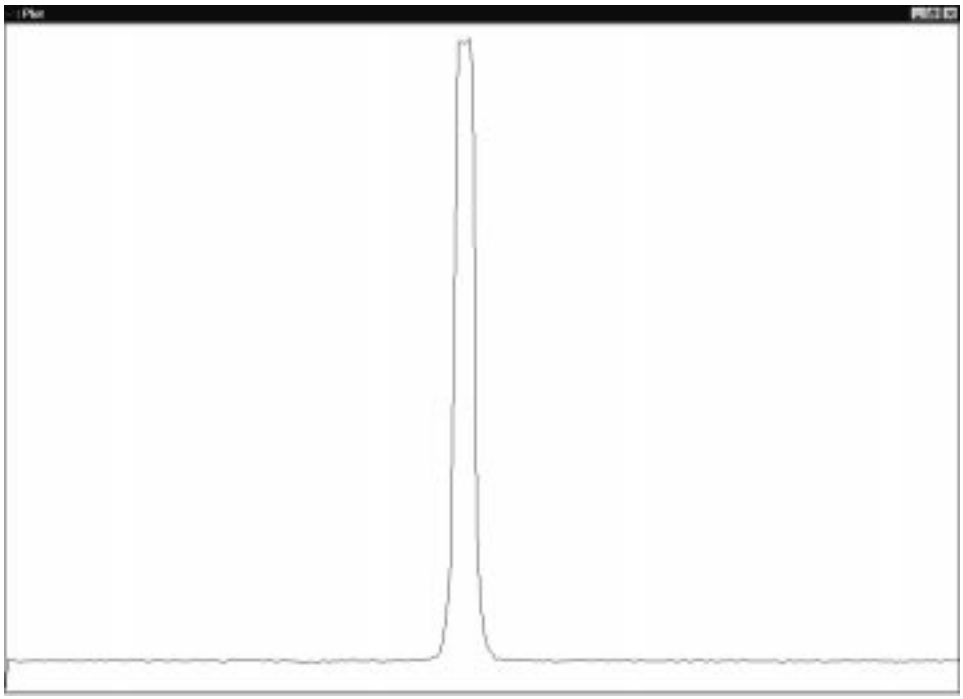

Figure 28

Analog Tuning

The following scope snapshot is from Pin 1 of U11. This is output from the first stage of our analog conditioning circuit. First, set the ground level of your scope so trace 1 is at the center line of the scope. Now, with DC coupling and triggering on channel 1 with negative slope, adjust R4 until the baseline of the video is on the scope center line. It should look like the figure below.

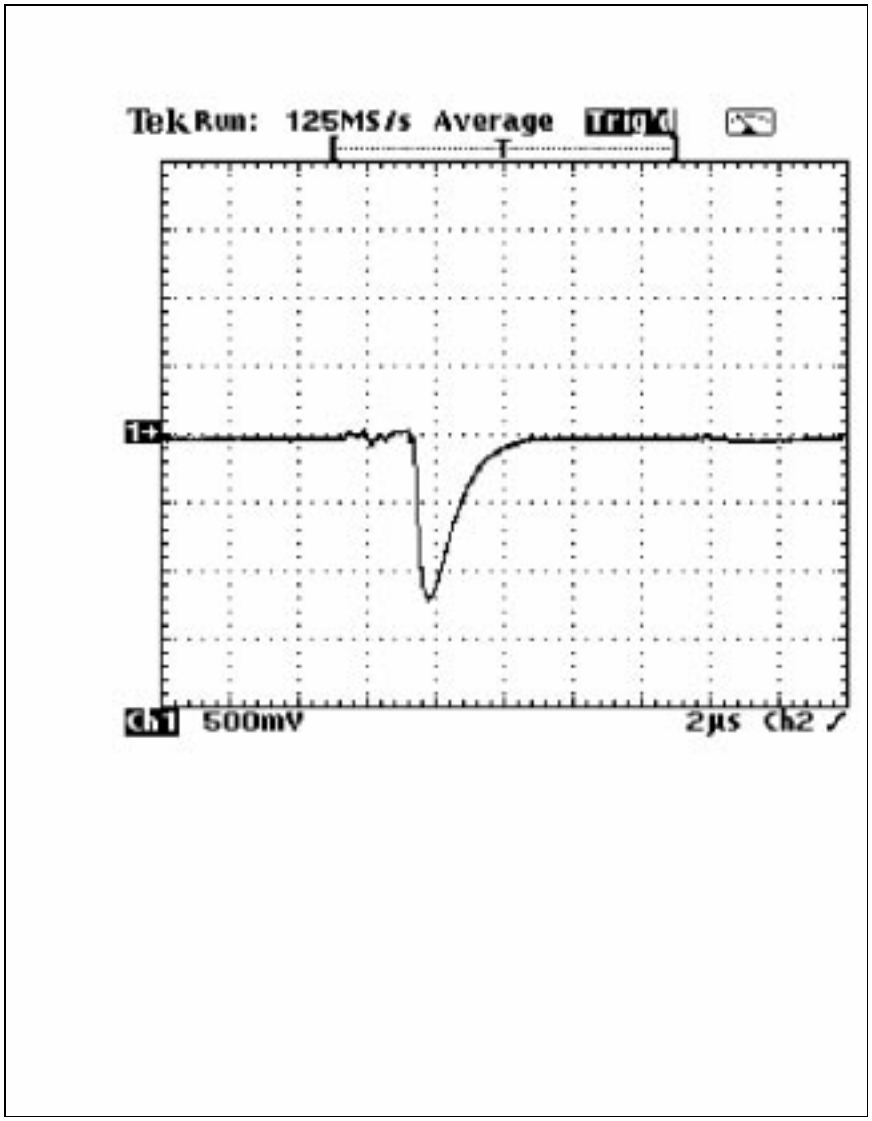

Figure 29

The following scope snapshot is from Pin 14 of U11. This is output from the fourth stage of our analog conditioning circuit. There are 2 adjustments here. Use R11 to adjust the bias of this signal. Move this signal up or down until its baseline is on the scope centerline. Now, adjust the gain of the signal using R12. You may have to re-adjust the bias again after adjusting gain via R12.

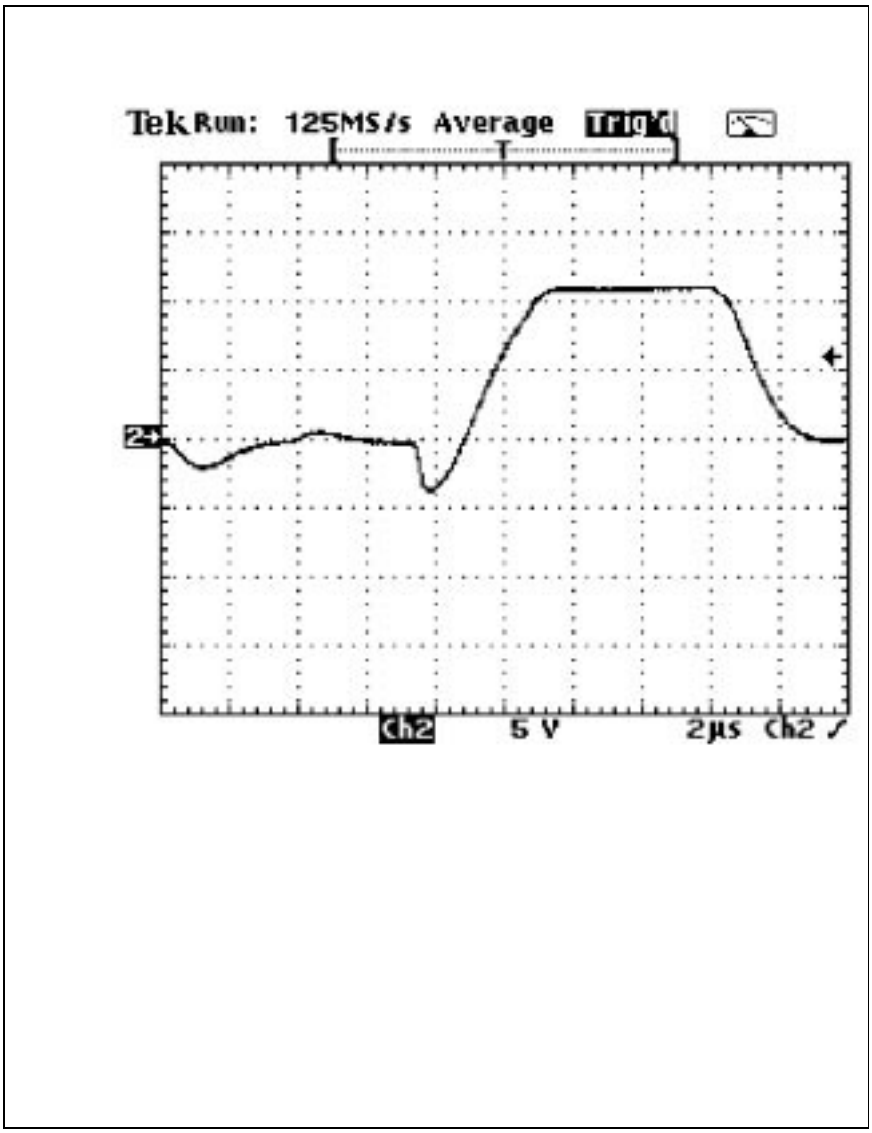

Figure 30

The following scope snapshot just shows the relationship between the first and fourth (last) stage of our analog conditioning circuit. Trace 1 is from Pin 1 of U11. Trace 2 is from Pin 14 of U11. Take notice of the gain change between our relatively low level video signal (trace 1) and our high level integrated video signal (trace 2).

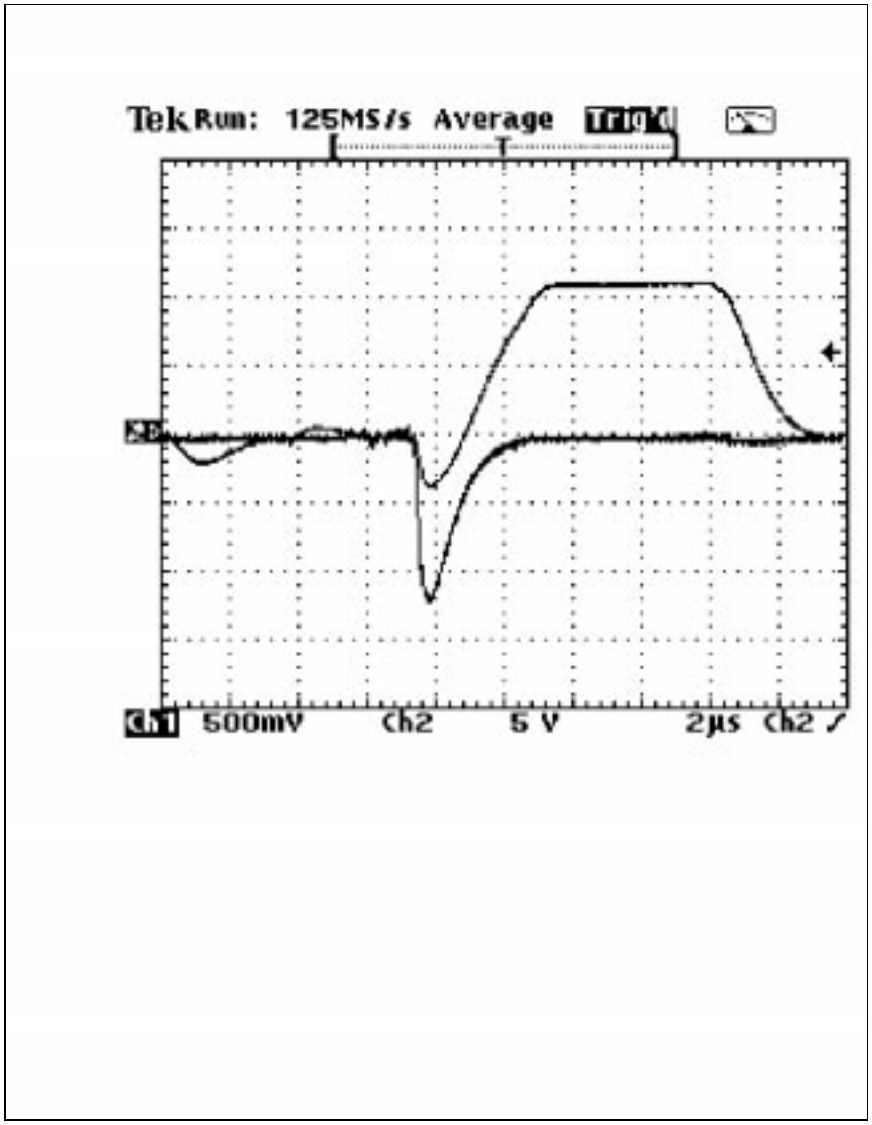

Figure 31

Printed Circuit Board

The following figure is the final circuit board drawn to scale. For convenience, I have included the top and bottom layers together. The 3 internal layers not seen are power, digital, and analog grounds.

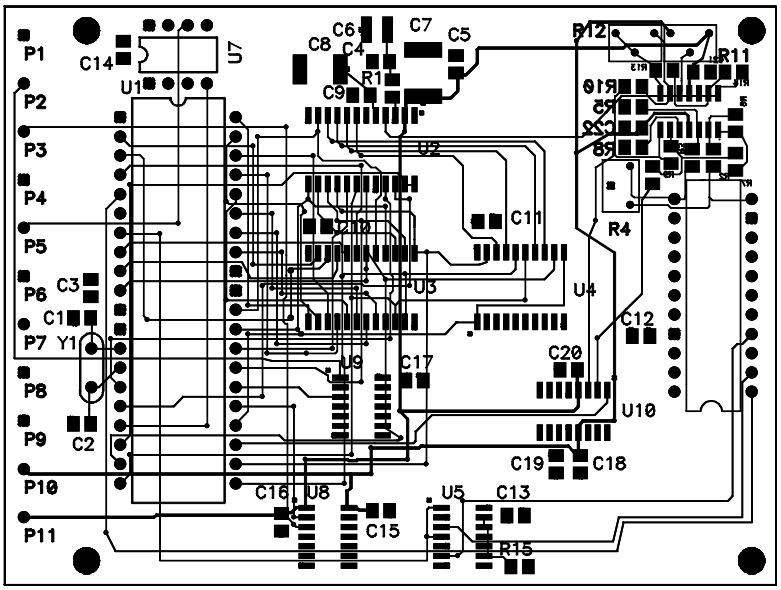

Figure 32

Spectral Output

The following figure is the spectral output of a fluorescent light taken with this new spectrometer board. This board was mounted and tested in the original chassis.

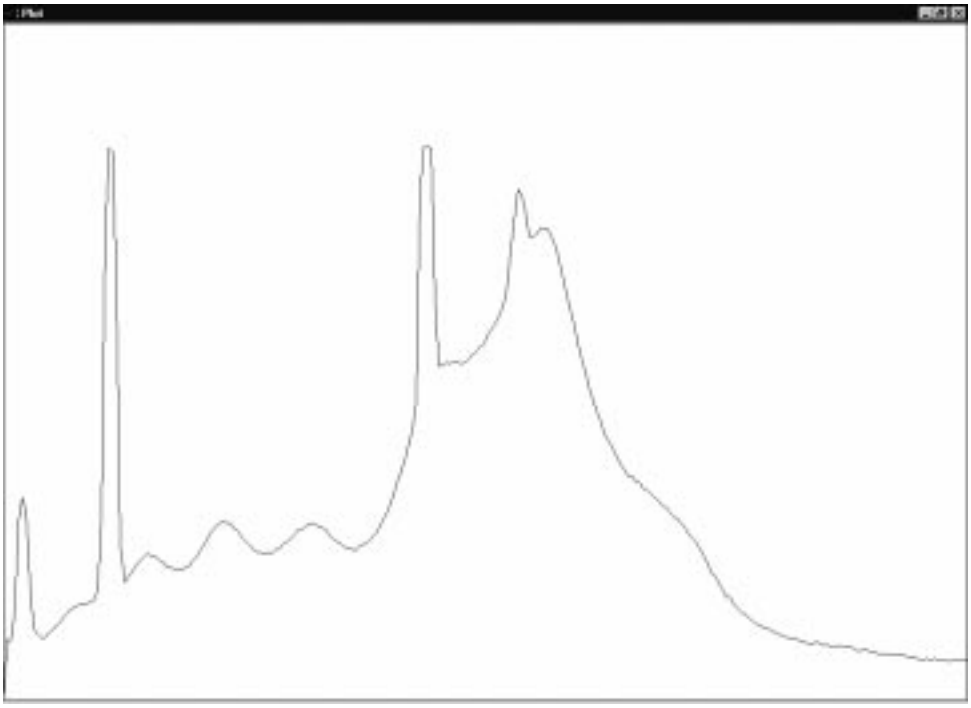

Figure 33